یک نمای کلی از تکنیک های تحمل پذیری خطا برای سیستم های زمان واقعی

چکیده

امروزه سیستم عامل‌ها بخشی جدایی‌ناپذیر سیستم‌های کامپیوتری هستند. سیستم عامل‌های زمان واقعی (RTOS) نوع خاصی از سیستم عامل هستند که هدف اصلی آنها درستی عمل و ارائه نتایج صحیح و معتبر در یک زمان محدود و از پیش تعیین شده است. RTOS به‌طور گسترده در حوزه ایمنی بحرانی استفاده می‌شود. در این حوزه تمام نیازهای سیستم باید مرتفع گردد و اگر سیستم نتواند یک فاجعه رخ می‌دهد. از این رو، تحمل‌پذیری خطا یک نیاز ضروری از RTOS به کار برده شده در حوزه ایمنی بحرانی است. در دهه‌های گذشته، تکنیک‌های تحمل‌پذیری خطا برای حفاظت از بخش‌های مختلف یک RTOS در برابر خطاها مطرح شده است. در این مقاله، پس از ارائه مفاهیم اساسی و RTOS، برخی از ویژگی‌های این سیستم عامل بررسی شده و پس از آن تعدادی از تکنیک‌های تحمل‌پذیری خطا که می‌تواند برای هر یک از ویژگی‌های کاربردی و تاثیر آنها بر قابلیت اطمینان سیستم به‌کار برده شود بررسی شده است. سهم اصلی از این کار بررسی و طبقه‌بندی تکنیک‌های مختلف تحمل‌پذیری خطا قابل اجرا در RTOS براساس ویژگی‌های سیستم عامل است.

کلمات کلیدی: سیستم عامل زمان واقعی، تحمل‌پذیری خطا.

1. مقدمه

"یک سیستم عامل به‌عنوان یک واسطه بین کاربر یک کامپیوتر و سخت‌افزار کامپیوتر عمل می‌کند. هدف از یک سیستم عامل ارائه یک محیط است که در آن کاربر می‌تواند برنامه را به شیوه‌ای مناسب و کارآمد اجرا کند"[1]. در

واقع نقش اصلی یک سیستم عامل به‌کار بردن برخی از روش‌های مدیریت سیستم کامپیوتر، مانند پردازنده برنامه‌ریزی، پردازش و مدیریت موضوع، ارتباط بین فرآیند، مدیریت حافظه، مدیریت I / O، کنترل همزمانی، بخش انتقادی، هماهنگ‌سازی، قطع و راه‌اندازی رویداد، کنترل ساعت و غیره است که به‌عنوان ویژگی‌های سیستم عامل شناخته شده است.

در جهان غیرزمان واقعی، دامنه مقادیر، بُعد محاسبات و صحت نتایج با شرط کافی برای مشخص کردن نتایج معتبر است. گنجاندن حوزه زمان در سیستم‌های زمان واقعی یک بُعد جدید به محاسبات می‌افزاید. برنامه‌های زمان واقعی علاوه بر تصحیح نتایج، باید به تولید نتایج معتبر نیز منجر شود. در این برنامه، صحت نتایج تولید شده و معتبر بودن آن‌ها در یک زمان محدود و از پیش تعیین شده به‌دست آمده است که نتایج درست در زمان تولید، به‌دست می‌آید.

سیستم عامل‌های اشتراک زمانی یک محیط برای اجرای برنامه ارائه می‌کنند و نتایج صحیح را با استفاده از منابع نسبتا کارآمد فراهم می‌کنند. کنترل RTOS معمولی و کنترل برخی از فرآیندهای خارجی/اشیاء، باید از تغییرات در پروسه خارجی/شی آگاه شده و به آنها به موقع پاسخ دهد. به‌منظور پاسخگویی به چنین محدودیت زمانی، RTOS با توجه به زمان واقعی مورد نیاز باید به موقع و قابل پیش‌بینی باشد درحالی‌که طراحی ویژگی‌های سیستم عامل که پیش از ذکر شد. در واقع، RTOS باید هر دو ویژگی مناسب و پیش‌بینی را برای توسعه نرم‌افزار فراهم کند.

یک سیستم به نام ایمنی بحرانی است اگر وقوع یک شکست در مواجه با نیازمندی‌های سیستم باعث اثرات فاجعه بار شود. علاوه بر نیازهای از پیش تعریف شده، این سیستم باید محدودیت‌های زمان واقعی را برآورده کند اگر آنها می‌خواهند به انجام وظایف در نظر گرفته شده خود به‌طور موثر بپردازند [2]. از این رو RTOS به‌طور گسترده در سیستم‌های ایمنی حساس استفاده می‌شود. هواپیماهای نظامی و غیر نظامی، نیروگاه‌های هسته‌ای و دستگاه‌های پزشکی نمونه‌هایی از سیستم ایمنی بحرانی هستند.

در سیستم‌های ایمنی مهم، علاوه بر سخت‌افزار، برنامه‌ها و سیستم عامل میزبان نیز باید مقاوم در برابر خطا و عملیات باشد. به‌عبارت دیگر، سیستم عامل به کار گرفته شده در حوزه ایمنی بحرانی باید نتایج صحیح و معتبری را در حضور و در غیاب خطا تولید کند. چنین ویژگی به‌عنوان محاسبات قابل اعتماد شناخته شده است [3]. نیازمندی‌های این

اعتبار برای اجرای تکنیک‌های تحمل‌پذیری خطا بر روی سیستم عامل است [4]. با وجود تلاش‌های انجام شده برای جلوگیری از حذف خطاها در مراحل توسعه سیستم‌های ایمنی بحرانی، خطای نرم‌افزار هنوز به‌طور کامل حذف نشده است و همچنین سخت‌افزار سیستم هنوز هم ممکن است در حین کار به‌دلیل خطای داخلی یا خارجی شکست بخورد. ازاین‌رو، اجرای تکنیک تحمل‌پذیری خطا در RTOS برای تحمل خطاها در یک سیستم ایمنی بحرانی بسیار مهم است. در این مقاله برای اولین بار برخی از مفاهیم اساسی RTOS ارائه شده و سپس تعدادی از مهم‌ترین ویژگی‌های RTOS بررسی می‌شود. پس از آن، برخی از تکنیک‌های تحمل‌پذیری خطای قابل اجرا با ویژگی‌های ذکر شده همراه با تاثیر آنها بر قابلیت اطمینان سیستم بررسی شده است. تکنیک‌های مورد بررسی شامل هر دو روش مبتنی بر نرم‌افزار-سخت‌افزار است و برای تحمل خطای گذرا و دائمی به‌کار برده می‌شود.

سازماندهی مقاله به شرح زیر است. در بخش 2 برخی مفاهیم اساسی و انواع مختلف RTOS ارائه شده است. بخش 3 به بررسی تعدادی از ویژگی‌های RTOS همراه با برخی از تکنیک‌های تحمل‌پذیری خطا است که می‌تواند برای هر یک از ویژگی‌ها استفاده شود. در نهایت در بخش 4 مقاله نتیجه‌گیری بیان شده است.

2. مفاهیم اساسی

در این بخش برای اولین بار برخی از تعاریف RTOS آورده شده است و سپس سه نوع از این سیستم عامل همراه با ملزومات اولیه مورد بحث ارائه شده است.

A) سیستم عامل زمان واقعی (RTOS)

"سیستم عامل‌های زمان واقعی برای پیش‌بینی، بهره‌وری استفاده می‌شوند و شامل ویژگی‌هایی برای حمایت محدودیت زمان هستند" [5]. در RTOS تمام وظایف باید به‌موقع (در زمان انتشار) منتشر شود و همچنین باید قبل از زمان خاص به نام مهلت به پایان برسد. وظیفه‌ی زمان واقعی اگر با شکست مواجه شود می‌تواند این محدودیت زمانی را برآورده نکند [6]. به‌عبارت دیگر، نقض محدودیت زمان‌بندی در RTOS منجر به شکست سیستم می‌شود. به‌منظور

تجزیه‌وتحلیل RTOS و گارانتی سیستم ایمنی، قطعات داخلی باید دقیقا تعریف شده و همچنین رفتار آنها باید قابل پیش‌بینی باشد.

به عنوان مثال XOberon یک RTOS کوچک است که قابلیت پیش‌بینی و ایمنی را همزمان فراهم می‌کند [7].

B) مهلت نرم، سفت و سخت

آخرین مهلت، ویژگی مهمی برای وظایف در RTOS زمانی که نتایج باید قبل از آن تولید شود است. اگر نتیجه پس از مهلت به‌دست آید، مهلت به‌عنوان نرم طبقه‌بندی می‌شود، در غیر این‌صورت سفت است. اگر عواقب شدیدی به‌وجود آید می‌تواند یک مهلت سفت از دست برود، در این‌صورت سخت نامیده می‌شود [8]. به‌عبارت دیگر نقض مهلت نتایج سفت در شکست و نقض مهلت نتایج سخت در فاجعه است.

نیازمندی‌های مختلف RTOS سخت و نرم نقش مهمی در طراحی سیستم دارد. اگر سیستم، محدودیت زمان واقعی سخت داشته باشد، طراح به‌صرف زمان زیادی برای گارانتی ایمنی و قابلیت پیش‌بینی و همچنین گارانتی همه‌ی محدودیت‌های زمانی (مهلت) نیاز خواهد داشت. در صورتی که شرایط زمان‌بندی سیستم نرم باشد، یک سیستم بهترین تلاش برای رسیدن به محدودیت زمان را دارد و همچنین دارای حداقل افت کیفیت است درحالی‌که نقض محدودیت زمان باید طراحی شود. بازیکنان رسانه قابل حمل و کنفرانس‌های ویدئویی آنلاین نمونه‌هایی از سیستم در زمان واقعی نرم هستند. نمونه‌هایی از سیستم‌های بلادرنگ سخت شامل سیستم درایوهای سیم در خودرو، سیستم پرواز با سیم در ارتباطات هوایی، سیستم‌های کنترل موشک و سیستم‌های فضایی مستقل هستند.

سیستم عامل‌های زمان واقعی سخت بر روی محدودیت‌های زمانبندی به‌عنوان مهم‌ترین مسئله در طراحی سیستم تمرکز دارند و نسبت به تحمل‌پذیری خطا به‌همان اندازه‌ی محدودیت زمان توجه ندارند. از آنجا که وقوع شکست در RTOS هم که باعث تولید نتایج نادرست به دلیل محاسبات اشتباه و یا باعث تولید نتایج نامعتبر می‌شود زیرا مهلت از دست رفته است، اجرای تکنیک تحمل‌پذیری خطا باید در طراحی سیستم در نظر گرفته شود. در این مقاله چند ویژگی

اصلی RTOS همراه با تعدادی از تکنیک‌های تحمل‌پذیری خطا ارائه شده است که می‌تواند در هر یک از ویژگی‌ها استفاده شود.

3. ویژگی RTOSS، و فنون تحمل‌پذیری خطا

در بخش‌های قبلی، به اهمیت پیاده‌سازی تکنیک‌های تحمل‌پذیری خطا در RTOS، به ویژه کسانی که در حوزه ایمنی بحرانی کار می‌کنند مورد بحث قرار گرفت. در این بخش، تعدادی از ویژگی‌های RTOS همراه با برخی از تکنیک‌های تحمل‌پذیری خطا ارائه شده است که می‌تواند برای هر یک از ویژگی‌ها استفاده شود.

A) مدیریت حافظه

به‌منظور حفاظت از اجزای سیستم عامل، مستعد ابتلا به شکست، تحمل‌پذیری خطا با حفاظت از حافظه آغاز می‌شود. زیرا رفتار برنامه به داده‌ها در حافظه بستگی دارد وجود خطا در این داده‌ها باعث خطا و شکست در برنامه می‌شود. از آنجا که انعطاف‌پذیری و قابلیت برنامه‌های کاربردی در حال افزایش است و همچنین آنها نیاز به دسترسی پویا به حافظه دارند، الگوریتم تخصیص ذخیره‌سازی پویا (DSA) نقش مهمی در سیستم عامل‌ها بازی می‌کند. علاوه بر انعطاف‌پذیری، برنامه‌های زمان واقعی نیاز به پیش‌بینی بیش از حد دارند، به‌عنوان مثال حافظه باید به صورت پویا در یک زمان محدود و از پیش تعیین شده اداره می شود. استفاده از DSA منجر به عدم قطعیت در RTOS، به دلیل زمان پاسخ نامحدود از الگوریتم DSA و مشکل تکه تکه شدن می‌شود. در [9] یک الگوریتم DSA با نام TLSF توسعه داده شده است که در RTOS به‌کار گرفته شده است. TLSF تخصیص صریح و تخصیص بلوک‌های حافظه با زمان‌بندی محدود و قابل قبول Θ (1) فراهم می‌کند. استفاده از بیت‌مپ ها و کمک بیت مپ روش دیگری برای تخصیص حافظه ایمن و قابل اعتماد است. این تکنیک توسط [10] که در RTEMS RTOS به کار گرفته شده است، معرفی می‌شود. سیستم عامل‌ها از واحد مدیریت حافظه (MMU) برای اجرای وظایف در آدرس حافظه محافظت شده استفاده می‌کنند. با این وجود برخی RTOS ها MMU و استفاده از آن را غیر فعال می‌کنند. [9] OSEK-VDX، µITRON و RTAI

نموه هایی از چنین RTOS هایی هستند که MMU را غیرفعال می‌کنند [11]. با غیرفعال کردن MMU سیستم عامل و تمام فرایندها در فضای آدرس همان را اجرا می‌کنند و هر وظیفه، دسترسی به کدها و داده‌های سیستم عامل و فرآیندهای دیگر را دارد. ازاین‌رو اگر یک کد بد نوشته شود و یا یک اشکال در یک کد باشد، برای مثال در اشاره‌گر مدیریت، منجر به شکست در هسته می‌شود و در نتیجه سقوط سیستم عامل می‌شود. بدون حفاظت آدرس حافظه، همچنین برخی از اشکالات خطا می‌شود که به سختی قابل تشخیص است. به‌عنوان مثال در پردازنده‌های PowerPC، RAM معمولا در آدرس فیزیکی 0 واقع می‌شود، طوری که حتی یک ارجاع اشاره‌گر NULL ممکن است شناسایی شود [12]. به‌منظور جلوگیری از چنین شکستی، RTOS باید MMU استفاده کند. با فعال کردن MMU، هر زمان که پشته یک کار سرریز کرد، یک استثنا سرریز مطرح شده است و سیستم عامل اجرای وظیفه را متوقف می‌کند. به‌جای متوقف کردن اجرای وظیفه، سیستم عامل می‌تواند کار را به حالت تعلیق درآورد و مشکل محدودیت پشته توسط مهاجرت وظیفه سرریز شده به یک فضای آدرس حافظه جدید با ظرفیت بزرگتر، با توجه فضاهای محفوظ و بی‌قید و شرط و سپس اجرای دوباره وظیفه‌ی معلق را حل کند. طراح RTOS باید زمان مهاجرت به‌هنگام تجزیه‌وتحلیل سیستم را داشته باشد.

رفع اشکالات یکی از تکنیک‌های مهم در تحمل‌پذیری خطا [3] است. این روش را می‌توان برای حافظه هنگامی‌که یک فرآیند بارگذاری می‌شود به‌کار برد، سیستم عامل داده‌های آن و حالت‌ها را در بیش از یک مکان/حافظه (سه مکان/ TMR) تکرار می‌کند. هر زمان که وظیفه/حالت یک داده تغییر می‌کند، این تغییرات به همه کپی‌ها اعمال می‌شود. هر زمان که کار می‌خواهد شروع به خواندن داده‌ها از حافظه کند، یک کپی برای تعیین اینکه آیا داده‌ها سهوا تغییر و یا خراب نشده‌اند (به هر دلیل، مانند تابش یونی سنگین) و همچنین برای تعیین داده درست می تواند مورد استفاده قرار گیرد. افزونگی حافظه می‌تواند در هر دو سطح نرم‌افزار و سطح سخت‌افزار [13] پشتیبانی شود.

علاوه بر افزونگی، یک سیستم مدیریت حافظه مقاوم در برابر خطا را می‌توان با چهار مکانیزم همزمان ساخت: اولین مکانیسم ضبط که برای ضبط به‌روزرسانی حافظه (نوشتن) حوادث فعال می‌شود، ثانیا مکانیسم ضبط دوم که سوابق حداقل تعداد محدودی از حافظه را ضبط می‌کند، یک فعال‌کننده برای فعال کردن اولین مکانیسم ضبط در این رویداد

از رخداد خطا و یک مکانیزم ادغام مجدد حافظه که توسط ادغام برخی از بخش‌هایی حافظه [14] برای بازیابی اطلاعات استفاده شده است. در این سیستم مدیریت حافظه، بازیابی خطا می‌تواند به سرعت و کارآمد با ادغام صفحات حافظه و با در نظر گرفتن به‌روزرسانی حافظه ورودی مشخص شده در مکانیزم ضبط اول و دوم انجام شود.

حافظه کد تصحیح خطا (حافظهECC) ابزاری برای بهبود قابلیت اطمینان سیستم عامل از منظر حفاظت از حافظه است. حافظه ECC یک نوع از ذخیره‌سازی داده‌های کامپیوتر است که توانایی تشخیص و تصحیح انواع بسیاری از خطاهای داخلی داده را دارد. این حافظه به خطاهای تک بیتی مقاوم است: داده‌ای که از هر کلمه خوانده می‌شود همان داده است که در آن نوشته شده است، حتی اگر یک بیت به حالت اشتباه [15] باشد. برخی از حافظه‌های غیر ECC با برابری پشتیبانی اجازه می‌دهد تا اشتباهات شناسایی شود، اما قابل اصلاح نیست. قابلیت اطمینان یک RTOS مقاوم در برابر خطا با به کارگیری این نوع از حافظه بهبود یافته است. در مقابل این تکنیک مبتنی بر سخت‌افزار، روش‌های مبتنی بر نرم‌افزار تشخیص و تصحیح خطا در حافظه وجود دارد که [16] هر دو قابلیت اطمینان و انعطاف‌پذیری را دارا هستند.

B) ملاحظات هسته

تشخیص خطا را می‌توان با روش‌های سخت‌افزاری و یا نرم‌افزار انجام داد، از جمله عنوان "تشخیص خطای گذرا از طریق چند رشته به‌طور همزمان" [17] که به‌عنوان مثال از روش‌های نرم‌افزاری است. هسته RTOS مقاوم در برابر خطا باید مکانیزمی را رائه کند که هر زمان که خطا رخ می‌دهد، یک پیام به یک عامل فرستاده شود که وظیفه انجام برخی از انواع اقدامات بازیابی خطا را فراهم می‌کند. این عامل که به نام سرپرست شناخته شده است باید در یک فضای آدرس جدا شده اجرا شود، زیرا داده‌ها در فضای آدرس حاوی وظیفه معیوب ممکن است خراب شوند. به‌عنوان مثال در Nooks که یک زیر سیستم قابلیت اطمینان است، مدیر بازیابی Nooks یک عامل برای بازیابی خطا [18] است. VxWorks RTOS یک ساختار درختی برای مدیریت اطلاع خطا توسط اجزای سطح بالاتر در درخت سلسله مراتبی [19] است که توسط قطعات سیستم عامل تولید شده است. سرپرست وظیفه‌ی بازیابی کارهای خطادار را با استفاده

از بهبود Forward -backward و یا شروع دوباره آن دارد. استراتژی بهبود انتخاب شده باید در تجزیه‌وتحلیل سیستم در نظر گرفته شده و تعریف شود.

هسته ملزم به ارائه یک مکانیسم ورود به سیستم برای تعیین ریشه خطا صریح با تجزیه‌وتحلیل همه چیز در سیستم است از قبیل تماس‌های خدمات کرنل، سوئیچ زمینه کار و وقفه، قبل از اتفاق افتادن خطا است [12]. برای تشخیص خطاهای ضمنی، کرنل باید قابلیت دیده‌بانی نرم افزار هنگامی که مطلع می‌شود داشته باشد هر زمان که یک وظیفه در توالی کد یا زمان برش انتظار آن اجرا می‌شود. این مکانیزم در روش چک کردن کنترل جریان [20] مفید است. به‌عنوان مثال QNX RTOS با استفاده از ماژول انتقادی فرآیند مانیتور (CPM) و VxWorks RTOS با استفاده از ماژول سیستم مدیریت غلبه بر خرابی (FMS) برای تشخیص اجزای سیستم درست عمل می‌کند.

به‌عنوان یک روش برای پیشگیری از خطا، تحمل‌پذیر خطای RTOS باید خود را در برابر سیستم فراخوانی نامناسب تماس و پارامترهای نامعتبر محافظت کند. به‌عنوان مثال برخی RTOS ها یک اشاره‌گر واقعی از اشیاء هسته (به‌عنوان مثال سمافور) به وظایف ارسال می‌کنند و زمانی که به دیگر تماس‌های خدمات کرنل ساخته شده توسط وظایف تغییر می‌یابند به این اشاره‌گر رجوع می کنند. در این رشته اگر یک کار پس از دریافت یک اشاره‌گر بیافتد، این شکست منجر به اشاره‌گرشده و در نتیجه باعث عبور از اشاره‌گر خراب در هسته می‌شود و با استفاده از آن RTOS ممکن است منجر به سقوط RTOS [12] شود. این نوع از شکست غیرممکن است، هسته RTOS باید پارامترهای ارسال به تمام تماس‌های خدمات اعتبار را داشته باشد. این اعتبار را می‌توان با به‌کارگیری توصیف برای مراجعه برنامه به هسته اشیاء و یا با استفاده از روش برنامه‌نویسی copy-N انجام داد [3].

در دسترس بودن بخش مهمی از محاسبات قابل اعتماد است که می‌تواند با ارائه تکرار گره عامل به‌دست آید. این کپی به صورت همزمان عمل می‌کند و داده‌ها و حالات درونی خود همزمان و هم معادل هستند. سیستم عامل شکست گره‌های ارسالی و دریافت پیام حیاتی از گره فعال را از طریق کانال‌های قابل اعتماد تشخیص می‌دهد. هنگامی که پیام حیات نتواند ببرسند، گره فعال دور انداخته می‌شود و یکی از گره‌های روی کار آمده به‌عنوان گره فعال مشخص شده

و پس از آن می‌توان آن را مورد پردازش قرار داد. شکل 1 این سناریو را به تصویر می‌کشد. در RTOS ترجیح می‌دهیم تا از تکرار فعال به جای تکرار منفعل با استفاده از تکنیک افزونگی استفاده کنیم [8].

RTOS مقاوم در برابر خطا نیز باید از گسترش گسل به هسته جلوگیری کند. این هدف را می‌توان با کاهش اندازه هسته و با نگه داشتن خدمات اساسی در داخل هسته و بدون در نظر گرفتن دیگران، به ویژه کسانی که در معرض خطا هستند، مانند رانندگان [21] به دست آورد. VxWorks RTOS این مرزهای حفاظت را با قرار دادن بین اجزای مختلف [19] مانند انزوا فراهم می‌کند.

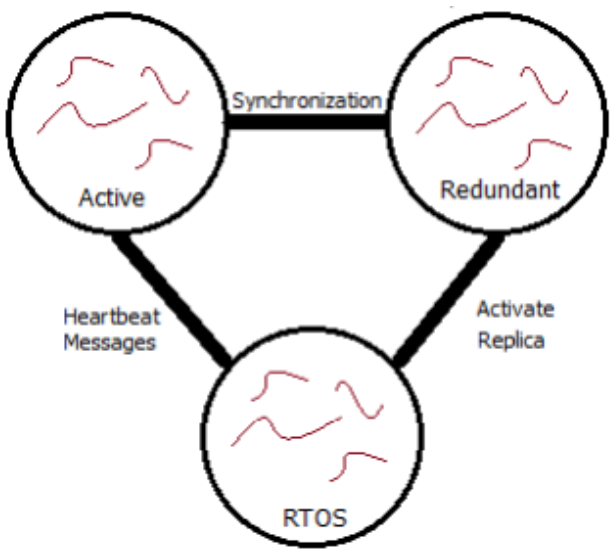

Figure 1 - Redundancy in Operating Nodes

C) فرایند و مدیریت موضوع

مشابه سیستم عامل‌های اشتراک زمانی، تعریف فرآیند و فعال‌سازی یکی از نقش‌های مهم RTOS است. اما، برخی از تفاوت‌های بین این دو نوع سیستم عامل در مدیریت فرآیندها به‌دلیل محدودیتهای زمانبندی در RTOSs وجود دارد. سیستم عامل‌های اشتراک زمانی بهترین تلاش خود را برای فعال شدن به‌موقع وظایف انجام می‌دهند. اما در مقابل، RTOS باید یک فرایند را یک بار فعال کند و آن را یک یا دوره انتشار دهد و همچنین گارانتی هر نسخه در زمان آغاز شده است و پیش از پایان مهلت آن به پایان می‌رسد. به‌منظور رعایت این محدودیت زمانی، یک RTOS باید در دسترس بودن منابع مورد نیاز فرآیندها را تضمین کند.

اگر رفتار وظایف توسط RTOS نظارت و کنترل نمی‌شود، یک کار ممکن است، به‌عنوان یک نتیجه مخرب و یا بی‌دقتی در کار، نمی‌تواند از پردازنده و یا دیگر منابع سیستم استفاده کند. زمانی که کار، یک کار جدید یا یک شیء دیگری از هسته ایجاد می‌کند، برخی از منابع سیستم، به ویژه یک تکه از حافظه و زمان پردازنده را به این کار جدید اختصاص می‌دهد. یک اشکال یا خطا در برنامه ممکن است منجر به وضعیتی شود که در آن این کار بسیاری از کارهای دیگر و یا اشیاء هسته و منابع سیستم را بیش از حد ایجاد کند. همانند نتیجه‌ی کارهای دیگر ممکن است به‌دلیل عدم توانایی خود را در کسب منابع مورد نیاز و در نتیجه اتمام مهلت شکست بخورد. در RTOS تحمل پذیری خطا، یک مکانیسم باید برای جلوگیری از چنین شکستی که ناشی از کمبود منابع است وجود داشته باشد. یک راه‌حل ممکن، تعیین حداکثر منابع مورد نیاز، به خصوص فضای حافظه و زمان CPU توسط فرآیندهای پیش از اتمام است، به‌طوری‌که RTOS می‌تواند منابع مورد نیاز برای هر فرآیند را رزرو کند و در نتیجه هیچ فرآیندی به‌دلیل کمبود منابع متوقف نشود. در این روش هیچ یک از فرآیندها نمی‌تواند بیش از منابع رزرو شده به‌دست آورد و اگر آنها بخواهند بیش از سهمیه خود استفاده کنند، این عمل به‌عنوان یک خطا در نظر گرفته و باید دور ریخته شود. از آنجا که در سیستم‌های با وظایف استاتیک، صفات همه وظایف از پیش شناخته شده است، یک رویکرد آسان می‌تواند در تخصیص منابع همانند اختصاص منابع RTOS به هر فرآیند از منابع رزرو شده برای آن و از منابع به جای مانده که رایگان هستند انتخاب شود و همچنین فرآیندهای دیگر محفوظ است. به‌عنوان مثال یک چارچوب به نام RRES [22] برای رزرو منابع معرفی شده است که با کمی برنامه نویسی بهبود قابل توجهی در قابلیت اطمینان سیستم به وجود خواهد آمد.

در سیستم‌های اولویت ثابت، اولویت وظایف به اشتباه به‌دلیل خطا در جدول فرآیند تغییر کرده است. روش‌های ممکن برای حل این مشکل آشنا کردن مدیریت فرآیند با اهمیت وظایف (به‌عنوان مثال کار سخت RT در مقابل کار RT نرم یا وظیفه انتقادی در مقابل وظایف عادی) با استفاده از پارتیشن در حافظه است. مفهوم مدیریت فرآیند پارتیشن بخش عمده‌ای از مشخصات ARINC 653، یک نرم‌افزار کاربردی رابط استاندارد [23] است. مدیریت فرآیند پارتیشن ARINC653 پارتیشن‌ها یا فضای آدرس را با توجه به جدول زمانی ارائه شده توسط طراح سیستم اجرا می‌کند. هر فضای آدرس در یک یا تعداد بیشتری پنجره اجرا قرار داده می‌شود. در طول هر پنجره، همه وظایف در فضای آدرس

دیگر نمی‌تواند اجرا شود، و تنها وظایف در فضای آدرس که درحال‌حاضر فعال هستند و توسط مدیر فرایند انتخاب شده‌اند اجرا می‌شود. هنگامی که پنجره فرآیندهای سخت/بحرانی فعال است، منابع پردازش آن تضمین شده است و فرآیندهای نرم/ عادی نمی‌توانند اجرا شوند و زمان پردازش از فرآیند سخت/ حیاتی گرفته می‌شود. یک پیاده‌سازی از 653ARINC در RTEMS RTOS نشان داده شده است [24].

D) برنامه‌ریزی

زمانبند، قلب یک RTOS است. در واقع به‌منظور حفظ ایمنی سیستم، زمانبند با درنظر گرفتن وظایف مجبور است تا تعیین کند چه کاری باید منتشر شود و باید در چه زمان قطع شود. الگوریتم برنامه‌ریزی‌های مختلفی در RTOS وجود دارد. مهم‌ترین آنها عبارتند از [25]:

RM : نرخ یکنواخت (RM) یک الگوریتم زمان‌بندی با اولویت ثابت است که وظایف اولویت تعریف شده است و وظایف با دوره کوچکتر تعریف اولویت بالاتری دارند.

EDF: فرآیند با کمترین زمان پایان ابتدا اجرا شد (EDF) یک الگوریتم زمان‌بندی پویا است که وظایف اولویت در زمان اجرا به‌صورت پویا تعریف شده که وظایف با مهلت نزدیک‌تر اولویت بالاتری دارند.

LLS: شبیه به EDF، حداقل سستی اول (LLF) یک الگوریتم زمان‌بندی پویا است. در این الگوریتم اختصاص اولویت براساس زمان سستی از یک فرآیند است. زمان Slack مقدار زمانی است که پس از یک کار در صورتی که کار در حال حاضر آغاز شده است سپری می‌شود. در LLS فرآیندهای با زمان Slack کوچکتر در اولویت بالاتری قرار دارند [26]. زمانبند به‌عنوان مهم‌ترین وظیفه از RTOS در برابر شکست محافظت می‌شود. اگر زمانبندی شکست بخورد، دیگر وظایف سیستم برنامه‌ریزی شده نیست و به‌درستی منتشر نمی‌شود و به‌عنوان نتیجه سیستم ار کار می‌افتد. اگر زمانبندی با اولویت ثابت باشد، سوء رفتار آن را می‌تواند با استفاده از یک جدول برنامه‌ریزی استاتیک از پیش ساخته و مقایسه خروجی زمانبندی با این جدول شناسایی مشخص کرد. این جدول باید برای یک دوره طولانی ساخته شود. برنامه‌نویسی N-copy (NCP) یکی دیگر از روش‌های تحمل‌پذیری خطا است که می‌تواند برای هر دو الگوریتم

برنامه‌ریزی با اولویت ثابت و پویا به کار برده شود. بااین روش، N نسخه از یک زمانبندی ($N \geq 3$) به‌صورت همزمان در فضای آدرس‌های مختلف اجرا می‌شود. سپس برنامه‌ریزی مناسب می‌تواند با در نظر گرفتن رای این کپی انجام شود.

علاوه بر تحمل‌پذیری خطا، زمانبندی باید زمان مورد نیاز برای انجام وظایف معیوب را تجزیه و تحلیل کند. همان‌طور که قبلا ذکر شد، یک کار معیوب می‌تواند دوباره آغاز شود و یا می‌تواند از آخرین بازرسی قبل از خطا دوباره بازسازی شود. این بازیابی و اجرای دوباره وظایف معیوب، زمان را هدر می‌دهد و می‌تواند محدودیت زمانی را نقض کند. به‌منظور ایمنی سیستم گارانتی در حضور شکست، RTOS مقاوم در برابر خطا باید این بار به‌هدر رفتن در تجزیه‌وتحلیل سیستم و طراحی زمانبندی را در نظر بگیرد. در تجزیه‌وتحلیل سیستم باید به صراحت برای هر وظیفه بسیاری از اجراهای دوباره امکان‌پذیر باشد، اگر شروع دوباره کار از ابتدا و در وضعیت استفاده از بهبود به‌عنوان طرح تحمل خطا، با داشتن تعیین نرخ، زمان تاخیر تشخیص خطا و زمان مورد نیاز برای صرفه‌جویی و بازگرداندن ایست‌های بازرسی داده‌ها انجام شود، در اغلب موارد چه تعداد از شکست‌ها می‌تواند بهبود یابد و همچنین چگونه بسیاری از پست‌های بازرسی برای انجام این کار باید گرفته شود [27، 28]. این تجزیه‌وتحلیل می‌تواند به‌صورت آماری در پیشرفت و یا به‌صورت پویا در زمان اجرا انجام شود. RTOS بیشتر تمایل به استفاده از state-G به جای ایست بازرسی دارد [8].

علاوه بر بازیابی وظایف از اشتباهات، RTOS مقاوم در برابر خطا باید قادر به بازیابی پردازنده از خطای گذرا و دائم باشد. اگر یک پردازنده به‌طور موقت با شکست مواجه شود حالات درونی خود و وظایف واگذار شده قابل بازیافت نیست، این شکست منجر به نقض محدودیت زمان و از کار افتادن سیستم می‌شود. با ارسال پیام‌های حیاتی، شکست پردازنده می‌تواند تشخیص داده شود و با داشتن پست‌های بازرسی کل پردازنده و وظایف محول شده، در دیسک، پردازنده معیوب از خطای گذرا عبور می‌کند [29]. در وضعیت خطای دائمی، پس از برگرداندن پردازنده معیوب، مهاجرت کار برای اجرای بهتر وظایف در پردازنده باید انجام شود. همچنین تحمل‌پذیری بیش از یک پردازنده معیوب ترجیحا باید در نظر طراحی سیستم [30] گرفته شده است. از آنجا که دیسک دارای سرعت پایینی است، استفاده از طرح واره‌های بدون دیسک و ذخیره‌سازی ایستگاه‌های بازرسی داده‌ها در حافظه دیگر پردازنده، هم به کاهش بار کمک می‌کند [31].

پژوهشی در اجرای زمانبندی تحمل‌پذیری خطا در RTEMS RTOS در [32] ارائه شده است. علاوه بر تحمل‌پذیری خطا، مدیریت انرژی و مسائل پوسته پوسته شدن ولتاژ پویا می‌تواند در زمان تجزیه‌وتحلیل به‌ویژه برای سیستم‌های جاسازی شده [33] در نظر گرفته شود.

E) ارتباطات

در تمام سیستم عامل‌ها، فرآیندهای نیاز به برقراری ارتباط با یکدیگر از طریق برخی مکانیسم‌ها، مانند عبور پیام و یا به اشتراک‌گذاری حافظه دارند. روش عبور پیام باعث عدم قطعیت، به‌دلیل ویژگی‌های معماری سیستم، در زمان سیستم می‌شود، به‌عنوان مثال تعیین این که دقیقا چه مدت عبور پیام طول می‌کشد غیرممکن است. در RTOS، حداکثر زمان تاخیر عبور پیام باید تعیین شود. برای رسیدن به چنین مقدار معینی برخی از تکنیک‌های مبتنی بر نشانه مانند حلقه و TDMA می‌تواند به‌کار گرفته شود [34]. علاوه‌براین اگر قابلیت اطمینان کانال‌های ارتباطی 100٪ نباشد، برخی از روش‌ها مانند افزونگی زمان پویا در سطوح پایین‌تر از پروتکل‌های ارتباطی و یا با استفاده از خدمات QoS را می‌توان برای افزایش قابل توجه قابلیت اطمینان کانال‌های ارتباطی به‌کار برد [34].

علاوه بر افزونگی زمانی فیزیکی و پویا، رویکردها و روش‌های دیگری برای افزایش قابلیت اطمینان بین فرآیندهای ارتباطات وجود دارد. به‌عنوان مثال در [35] یک مرکز معرفی شده است که پشتیبانی فرآیند مقاوم در برابر خطا توسط یک خانواده از پروتکل چندپخشی قابل اعتماد را که می‌تواند در RTOS مقاوم در برابر خطا استفاده شود فراهم می‌کند. در این مرکز یک پروتکل که ضامن ترتیب تحویل معرفی شده است تضمین کننده‌ی فرآیندهای متعلق به یک گروه از فرآیند مقاوم در برابر خطا است که قابلیت اطمینان سیستم از جمله شکست فرآیند، بهبود، مهاجرت را تحت تاثیر قرار می‌دهد و تغییرات پویایی در ویژگی‌های گروه مانند رتبه‌بندی انجمن‌ها دارد. که با استفاده از برخی از شکل‌های هندسی اولیه پخش، مانند: پخش گروه (GBCAST)، پخش اتمی (ABCAST) و پخش عادی (CBCAST) انجام می‌شود.

مشابه ارتباطات بین فرآیندها، ب ارتباطات ین پردازنده‌ها باید بیش از حد قابل اعتماد است. در [36] یک زیرسیستم به‌نام Transis معرفی شده است که بااستفاده از خدمات چندپخشی و قابل‌اعتماد پیام، از ارتباطات قابل‌اعتماد در میان پردازنده‌ها پشتیبانی می‌کند. VxFusion پسوند زمان اجرا برای حمایت از ارتباط بین پردازنده است که توسط VxWorks RTOS به‌کار برده شده است [19] . علاوه بر مکانیسم ذکر شده، مدل‌های کانال مختلف وجود دارد که بااستفاده از نرم‌افزارهای تبدیل کننده و رمزگشای مناسب، قابلیت اطمینان کانال‌ها را تضمین می‌کند [37]. به‌منظور انتخاب یک مدل برای یک ارتباط، عوامل متعددی باید در نظر گرفته شود. این عوامل عبارتند از ماهیت فیزیکی و آماری اختلالات کانال، اطلاعات در دسترس فرستنده و گیرنده، حضور هر لینک بازخورد از گیرنده به فرستنده، و در دسترس بودن فرستنده و گیرنده از یک منبع به اشتراک گذاشته شده (مستقل از اختلالات کانال) [37].

RPC یک روش ارتباطی راه دور است که به‌منظور دیدار با الزامات RTOS تحمل‌پذیری خطا، باید به شیوه‌ای قابل اعتماد انجام شود. Sun Batching در RPC یک تنوع از RPC است که انجام ارتباطات از راه دور را قابل اعتماد می‌کند. به‌طور معمول از پروتکل‌های جریان بایت قابل اعتماد (مانند TCP) برای حمل‌ونقل آن [38] استفاده می‌کند که علاوه بر گارانتی ارتباطات قابل‌اعتماد، ضمانت تحویل پیام را نیز داراست. این ویژگی‌ها منجر به Sun Batching در RPC توسطRTOS مقاوم در برابر خطا می‌شود.

اجرای یک RPC تحمل‌پذیری خطا مبتنی بر برنامه‌های کاربردی شبکه در [39] مورد بحث قرار گرفته است.

F) مدیریت I / O

RTOS باید دسترسی I / O را مدیریت کنند که از دخالت‌ها پیشگیری شود و همچنین تمام وظایف (به‌خصوص وظایف زمان واقعی سخت) می‌تواند محدودیت‌های زمان‌بندی خود را تامین کند. علاوه بر توجه به محدودیت زمان، تحمل‌پذیری خطا RTOS باید برخی از تکنیک‌های تحمل‌پذیری خطا را برای تحمل‌پذیری دستگاه‌های I / O معیوب فراهم کند. بسیاری از تکنیک‌های تحمل‌پذیری خطا برای دستگاه‌های I / O مربوط به دستگاه مورد نظر وجود دارد. تکرار روش متداولی است که می‌تواند با تکثیر دستگاه‌های I/ O به‌کار برده شود. دستگاه اصلی I / O فعال (اولیه)

نامیده می‌شود و آنهایی که تکرار می‌شوند پشتیبان‌گیری نامیده می‌شود. هنگامی که یک دستگاه فعال با شکست مواجه می‌شود و شکست آن توسط پیام حیاتی شناسایی شده است، یکی از دستگاه‌های پشتیبان باید وظایف دستگاه فعال از نقطه خطا را انجام دهد. چنین تکراری از طریق PCI در [40] بررسی شده است. به‌منظور کاهش اتلاف بار، از آن برای طراحی پشتیبان‌گیری به عنوان افزونگی فعال استفاده می‌شود. به‌عنوان مثال RAID یک مثال از افزونگی فعال در دستگاه‌های ذخیره‌سازی ثانویه [41] است.

استحکام یک ویژگی مهم کیفیت سیستم است که توسط واژه‌نامه استاندارد IEEE در اصطلاحات مهندسی نرم‌افزار به‌عنوان: "درجه‌ای که یک سیستم و یا جزء می‌تواند به‌درستی در حضور ورودی نامعتبر و یا شرایط استرس زا از محیط زیست تابع عمل کند" تعریف شده است [42]. Avizienis و همکارانش استحکام را به‌عنوان "قابلیت اعتماد با توجه به ورودی نادرست» [43] تعریف کرده‌اند. هنگامی که داده‌های ورودی از دست می‌روند یا نادرست هستند، تکنیک‌های نیرومندی سعی در رفع و یا محاسبه مقدار دقیق یا تقریبی از داده‌های ورودی می‌کنند. یک روش در استحکام درخواست اطلاعات صحیح از فرستنده کاربر با در نظر گرفتن فرمت داده‌های درست است که در یک جدول الگوی داده از پیش تعریف شده، تعریف شده است. روش دیگر استفاده از آخرین اطلاعات صحیح به جای از دست رفته / داده‌های ورودی نادرست رسیده و یا تقریبی صحیح از داده‌های ورودی با استفاده از برخی الگوریتم‌های یادگیری ماشین به داده‌های ورودی قبلی در موقعیت‌های مشابه است. چنین تکنیک‌هایی یک رفتار صحیح در سیستم را تضمین نمی‌کنند، اما آنها عوارض جانبی از دست دادن داده را کاهش می‌دهند.

استحکام سیستم‌عامل توسط توانایی رابط‌های برنامه کاربردی در دست زدن به پارامترهای ورودی استثنایی که متشکل از تشخیص پارامترهای نامعتبر و تحمل آنها [44] است اندازه‌گیری می‌شود. آزمایش بر روی 233 تابع از POSIX 13 یک استحکام میزان شکست از 6٪ تا 19٪ برای آزمایش تک سیستم عامل است که با استفاده از روش copy-N این میزان را به 3.8٪ کاهش دادیم.

مدل نیرومندی مورد نظر باید همزمان با توسعه سیستم انتخاب شود. براساس مطالعه‌ای در [45]، تقریبا 47 درصد از تحقیقات در نظر نیرومندی برای تأیید و اعتبارسنجی مرحله‌ای از توسعه سیستم و فقط 35 درصد از تحقیقات آن در فاز طراحی سیستم در نظر گرفته شده است. تحقیقات دیگر آن را در مراحل مختلفی انجام داده‌اند.

G) بررسی وقفه

سیستم‌عامل‌ها انواع مختلفی از وقفه با اولویت‌های مختلف و زمان اجرا دارند. وقفه با اولویت بالاتر نیاز به یک زمان پاسخ‌دهی سریع‌تر دارد. هنگامی که ساختارهای داده داخلی در طول تماس‌های خدمات دستکاری می‌شوند، دیگر وقفه‌ها به‌ویژه زمان‌بندی تایمر باید غیرفعال باشد زیرا در غیر این‌صورت یک تماس خدمات مرتبط ممکن است اجرا و باعث دسترسی به داده‌های متناقض شود. در واقع به‌منظور رسیدگی به اولویت‌های پایین‌تر قابل اعتماد، وقفه اولویت بالاتر به دور انداخته و یا مانع می‌شود که باعث عدم قطعیت در زمان تماس‌های ناخواسته سیستم برای RTOS است [12]. روش تجزیه‌وتحلیل استاتیک برای به‌دست آوردن WCET از تماس‌های سیستم در RTEMS RTOS در [46] معرفی شده است.

RTOS مقاوم در برابر خطا باید هر دو پیش‌بینی و قابلیت اطمینان را تضمین کند درحالی‌که وقفه را اعمال می‌کند. برای دستیابی به این اهداف، همه تماس‌های خدمات کرنل باید revertible شود، به‌طوری‌که RTOS می‌تواند به تماس خدمات پیش‌دستی کرده وموجب بازگرداندن عملیات انجام داده شده بعد از راه‌اندازی مجدد آن شود. بنابراین زمان برای بازگشت به زمانبندی ممکن است با چند دستورالعمل و وقفه اولویت بالاتر همیشه با حداقل تاخیر مطلق اجرا شده باشد. این روش قابلیت پیش‌بینی و قابلیت اطمینان سیستم را از نظر دسترسی به داده‌های متناقض بهبود می‌بخشد.

H) زبان‌های برنامه‌نویسی

از آنجا که RTOS مقاوم در برابر خطا شرایط خاصی دارد، به‌منظور مواجه با آنها زبان‌های برنامه‌نویسی ویژه‌ای باید به‌خوبی استفاده شود. برخی از ویژگی‌های زبان‌های برنامه‌نویسی سنتی در معرض ابتلا به مشکلاتی با استفاده از آنها در تحمل‌پذیری خطا RTOS هستند، مانند: اشاره‌گرها، تخصیص حافظه پویا و آزادسازی، برنامه‌نویسی بدون ساختار، نقاط ورود متعدد و نقاط خروج، داده‌های مختلف، اعلام ضمنی و مقداردهی اولیه ضمنی، پارامترهای رویه، بازگشت، همزمانی و قطع برنامه‌نویسی [47]. علاوه بر توجه به این ویژگی‌های برنامه‌نویسی، برنامه‌های زمان واقعی پاسخ‌های درست درون محدودیت‌های زمان‌بندی دقیق را تضمین می‌کنند. به‌عبارت دیگر حداکثر زمان مورد نیاز برای پاسخ به یک درخواست یا برای تکمیل کار توسط یک فرآیند باید پاسخگو باشد. برای رسیدن به این زمان، حداکثر زمان که هر بخشی از یک برنامه طول می‌کشد باید به صراحت مشخص شود. از این رو به‌عنوان مثال در برنامه‌نویسی زمان واقعی، حلقه متغیر با تکرار نامشخص و یا نامحدود قابل قبول نیست.

به‌طورکلی، زبان‌های برنامه نویسی در زمان واقعی در سه مدل برنامه‌نویسی زمان واقعی به عنوان سنکرون، برنامه‌ریزی شده، و به پایان رسیده است که در زمان آنها برای کامل و کامپایل شدن متفاوت است [48]. علاوه بر توجه به این مدل‌ها، زبان‌های برنامه‌نویسی مورد استفاده در RTOS مقاوم در برابر خطا باید برخی از تکنیک‌های تشخیص خطا و تصحیح خطا را پشتیبانی کند. آدا یکی از گسترده‌ترین زبان‌های برنامه نویسی در تحمل‌پذیری خطا در زمان واقعی به‌دلیل نقاط قوت آن مانند: معناشناسی تعریف شده، کنترل نوع قوی، مکانیزم ساختار مانند بسته‌بندی و حمایت از توسعه تجزیه‌وتحلیل کد، تأیید و ابزار تست است [49]. Euclid یکی دیگر از زبان‌های برنامه‌نویسی زمان واقعی مقاوم در برابر خطا که از گرداننده استثناها و لیست ورودی‌ها / خروجی‌ها برای ارائه جامع تشخیص خطا، انزوا، و بهبود استفاده می‌کند. فلسفه این زبان این است که هر استثنا توسط سخت‌افزار و یا نرم‌افزار قابل تشخیص است. علاوه‌براین، Euclid همه چیز را در زبان به زمان و فضای محدود تحمیل می‌کند [50]. استفاده از چنین زبان برنامه‌نویسی باعث بهبود قابلیت اطمینان سیستم می‌شود[51].

4. نتیجه‌گیری

سیستم‌عامل زمان واقعی به‌طور گسترده در حوزه ایمنی بحرانی برای تعامل با اشیاء کنترل شده در محیط خارجی استفاده می‌شود و نتایج صحیح و معتبر در یک زمان محدود و از پیش تعیین شده ارائه می‌کند. در این حوزه، هزینه‌های شکست سیستم منجر به فاجعه و بیش از سرمایه‌گذاری اولیه در کامپیوتر و در جسم کنترل شده می‌شود. برای جلوگیری از چنین شکستی، طراح سیستم باید تضمین کند که سیستم می‌تواند نیازهای مشخص شده در حوزه‌های مقدار و زمان را در طول تمام شرایط عملیاتی پیش‌بینی شده، حتی زمانی که خطا رخ می‌دهد تشخیص دهد. برای رسیدن به این هدف، RTOS به‌کار برده شده باید قادر به تحمل خطا و خطاهای موجود در سیستم باشد.

شبیه سیستم عامل سنتی، RTOS از ویژگی‌های بدوی برای یک RTOS عمومی و برای پاسخگویی به مقدار و زمان مورد نیاز است. پیاده‌سازی تکنیک‌های تحمل‌پذیری خطا در این ویژگی موب بهبود قابلیت اطمینان RTOS و کل سیستم می‌شود.

در این مقاله برای اولین بار برخی از تعاریف RTOS را همراه با الزامات خود و با بررسی برخی از ویژگی‌های ابتدایی RTOS مانند مدیریت حافظه، ملاحظات هسته، فرآیند و موضوع مدیریت، ارتباطات، مدیریتI / O ، مدیریت وقفه و زبان‌های برنامه‌نویسی به‌دنبال داشت. سپس تعدادی از تکنیک‌های تحمل‌پذیری خطا که می‌تواند به هر یک از ویژگی‌های ذکر شده اعمال شود ارائه شده است. این مقاله در واقع چندین تکنیک تحمل‌پذیری خطا قابل اجرا در RTOS براساس برخی از ویژگی‌های ابتدایی از سیستم عامل‌های طبقه‌بندی است. برخی از تکنیک‌ها تنها می‌تواند با خطای گذرا مقابله کند و برخی می‌تواند هر دو خطای گذرا و دائمی را تحمل کند. برخی از تکنیک‌های تنها براساس نرم‌افزار است و برخی به سخت‌افزار تکیه می‌کند. به‌منظور RTOS تحمل‌پذیری خطا، طراح سیستم باید نیازمندی‌های تکنیک‌های تحمل‌پذیری خطا را در تحلیل نیازمندی‌ها و فاز طراحی سیستم درحالی‌که سیستم درحال توسعه است در نظر بگیرد.

REFERENCES

[1] A. Silberschatz, P. B. Galvin, and G. Gagne, Operating system concepts: J. Wiley & Sons, 2009.

[2] J. S. Ostroff, "Formal methods for the specification and design of real-time safety critical systems," Journal of Systems and Software, vol. 18, pp. 33-60, 1992.

[3] L. L. Pullum, Software fault tolerance techniques and implementation: Artech House Publishers, 2001.

[4] P. J. Denning, "Fault tolerant operating systems," ACM Computing Surveys (CSUR), vol. 8, pp. 359-389, 1976.

[5] J. A. Stankovic and R. Rajkumar, "Real-time operating systems," Real-Time Systems, vol. 28, pp. 237-253, 2004.

[6] P. A. Laplante, "Real-Time Systems Design and Analysis," 1993.

[7] R. Brega, "A real-time operating system designed for predictability and runtime safety," in Proceedings of The Fourth International Conference on Motion and Vibration Control (MOVIC), 1998, pp. 379-384.

[8] H. Kopetz, Real-time systems: design principles for distributed embedded applications vol. 25: Springer, 2011.

[9] M. Masmano, I. Ripoll, A. Crespo, and J. Real, "TLSF: A new dynamic memory allocator for real-time systems," in Real-Time Systems, 2004. ECRTS 2004. Proceedings. 16th Euromicro Conference on, 2004, pp. 79-88.

[10] H. Li and C. Yin, "Analysis and Improvement of RTEMS Memory Management," in Education Technology and Computer Science, 2009. ETCS'09. First International Workshop on, 2009, pp. 107-111.

[11] R. Yerraballi, "Real-time operating systems: An ongoing review," in Proceedings of the 21st IEEE Real-Time Systems Symposium (RTSS'2000), WIP Section, Orlando Fl, 2000.

[12] David Kleidermacher and M. Griglock, "Real-time Operating System Requirements for Use in Safety Critical Systems," Green Hills Software, Inc2001.

[13] K. S. Gray, "Memory redundancy techniques," ed: Google Patents, 2002.

[14] E. J. Williams, "Memory management in fault tolerant computer systems," ed: EP Patent 0,817,053, 2003.

[15] F. Qin, S. Lu, and Y. Zhou, "Safemem: Exploiting ECC-memory for detecting memory leaks and memory corruption during production runs," in HighPerformance Computer Architecture, 2005. HPCA-11. 11th International Symposium on, 2005, pp. 291-302.

[16] C. Borchert, H. Schirmeier, and O. Spinczyk, "Generative software-based memory error detection and correction for operating system data structures," in Dependable Systems and Networks (DSN), 2013 43rd Annual IEEE/IFIP International Conference on, 2013, pp. 1-12.

[17] S. K. Reinhardt and S. S. Mukherjee, "Transient fault detection via simultaneous multithreading," in ACM SIGARCH Computer Architecture News, 2000, pp. 25-36.

[18] M. M. Swift, B. N. Bershad, and H. M. Levy, "Improving the reliability of commodity operating systems," ACM Transactions on Computer Systems (TOCS), vol. 23, pp. 77-110, 2005.

[19] A. K. Sood, "Digging Inside the VxWorks OS and Firmware (The Holistic Security)," SecNiche Security Labs.

[20] I. Koren and C. M. Krishna, Fault-tolerant systems: Morgan Kaufmann, 2010.

[21] J. N. Herder, H. Bos, B. Gras, P. Homburg, and A. S. Tanenbaum, "Construction of a highly dependable operating system," in Dependable Computing Conference, 2006. EDCC'06. Sixth European, 2006, pp. 3-12.

[22] A. Mancina, D. Faggioli, G. Lipari, J. N. Herder, B. Gras, and A. S. Tanenbaum, "Enhancing a dependable multiserver operating system with temporal protection via resource reservations," Real-Time Systems, vol. 43, pp. 177-210, 2009.

[23] A. Specification, "653," Avionics Application Software Interface, Annapolis, MD, 1997.

[24] J. Rufino, S. Filipe, M. Coutinho, S. Santos, and J. Windsor, "ARINC 653 interface in RTEMS," in Proc. DASIA, 2007.

[25] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," Journal of the ACM (JACM), vol. 20, pp. 46-61, 1973.

[26] M. L. Dertouzos and A. K. Mok, "Multiprocessor online scheduling of hardreal-time tasks," Software Engineering, IEEE Transactions on, vol. 15, pp. 1497-1506, 1989.
[27] Y. Zhang and K. Chakrabarty, "A unified approach for fault tolerance and dynamic power management in fixed-priority real-time embedded systems," Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, vol. 25, pp. 111-125, 2006.
[28] I. J. Bate, "Scheduling and timing analysis for safety critical real-time systems," Ph.D., University of York Department Of Computer SciencePublications-Ycst, 1999.
[29] G. Bournoutian and A. Orailoglu, "Dynamic transient fault detection and recovery for embedded processor datapaths," in Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, 2012, pp. 43-52.
[30] X. Ping and Z. Xingshe, "Security-Driven Fault Tolerant Scheduling Algorithm for High Dependable Distributed Real-Time System," in Parallel Architectures, Algorithms and Programming (PAAP), 2011 Fourth International Symposium on, 2011, pp. 29-33.
[31] G.-M. Chiu and J.-F. Chiu, "A new diskless checkpointing approach for multiple processor failures," Dependable and Secure Computing, IEEE Transactions on, vol. 8, pp. 481-493, 2011.
[32] B. Zhang, X. Xu, and B. Li, "Research on the design of software fault tolerance based on RTEMS," in Computer, Mechatronics, Control and Electronic Engineering (CMCE), 2010 International Conference on, 2010, pp. 402-405.
[33] T. Wei, P. Mishra, K. Wu, and J. Zhou, "Quasi-static fault-tolerant scheduling schemes for energy-efficient hard real-time systems," Journal of Systems and Software, vol. 85, pp. 1386-1399, 2012.
[34] A. S. Tanenbaum, Modern operating systems vol. 2, 1992.
[35] K. P. Birman and T. A. Joseph, "Reliable communication in the presence of failures," ACM Transactions on Computer Systems (TOCS), vol. 5, pp. 47-76, 1987.
[36] Y. Amir, D. Dolev, S. Kramer, and D. Malki, "Transis: A communication subsystem for high availability," in Fault-Tolerant Computing, 1992. FTCS-22. Digest of Papers., Twenty-Second International Symposium on, 1992, pp. 76- 84.
[37] A. Lapidoth and P. Narayan, "Reliable communication under channel uncertainty," Information Theory, IEEE Transactions on, vol. 44, pp. 2148- 2177, 1998.
[38] R. Thurlow, "RPC: Remote procedure call protocol specification version 2," 2009.
[39] Y. Tanimura, T. Ikegami, H. Nakada, Y. Tanaka, and S. Sekiguchi, "Implementation of fault-tolerant GridRPC applications," Journal of Grid Computing, vol. 4, pp. 145-157, 2006.
[40] S. L. Blinick, J. C. Elliott, and E. Q. Garcia, "Redundant and fault tolerant control of an I/O enclosure by multiple hosts," ed: Google Patents, 2011.
[41] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson, "RAID: High-performance, reliable secondary storage," ACM Computing Surveys (CSUR), vol. 26, pp. 145-185, 1994.
[42] J. Radatz, A. Geraci, and F. Katki, "IEEE standard glossary of software engineering terminology," IEEE Std, vol. 610121990, p. 121990, 1990.
[43] A. Avizienis, J.-C. Laprie, and B. Randell, Fundamental concepts of dependability: University of Newcastle upon Tyne, Computing Science, 2001.
[44] P. Koopman and J. DeVale, "Comparing the robustness of POSIX operating systems," in Fault-Tolerant Computing, 1999. Digest of Papers. Twenty-Ninth Annual International Symposium on, 1999, pp. 30-37.
[45] A. Shahrokni and R. Feldt, "A systematic review of software robustness," Information and Software Technology, 2012.
[46] A. Colin and I. Puaut, "Worst-case execution time analysis of the RTEMS real-time operating system," in Real-Time Systems, 13th Euromicro Conference on, 2001., 2001, pp. 191-198.
[47] I. I. P. Ltd., "An Introduction to Safety Critical Systems," 1997.
[48] C. M. Kirsch, "Principles of real-time programming," in Embedded Software, 2002, pp. 61-75.
[49] T. S. Taft and R. A. Duff, Ada 95 Reference Manual. Language and Standard Libraries: International Standard ISO/IEC 8652: 1995 (E) vol. 1246: Springer, 1997.

[50] E. Kligerman and A. D. Stoyenko, "Real-time Euclid: A language for reliable real-time systems," Software Engineering, IEEE Transactions on, pp. 941-949, 1986.
[51] V. Barr and S. Montenegro, "BOSS/Ada: An Open Source Ada 95 Safety Kit A Dependable open source embedded operating system for GNAT," Ada Deutschland Tagung, pp. 53-66, 2002.