دومین کنفرانس ملی اویونیک ایران
۱۴ و ۱۵ بهمن ماه ۱۳۹۳
دانشگاه صنعتی اصفهان

مدیریت در زمان اجرا حافظه SPM در سطح سیستم عامل میکروسی

محمد حسین کیانی[1]، محمد علی منتظری[2] و علی بهلولی[3]
[1] دانشجو کارشناسی‌ارشد مهندسی کامپیوتر، دانشگاه صنعتی اصفهان، اصفهان، mohammad.kiani@ec.iut.ac.ir
[2] استادیار دانشکده مهندسی برق و کامپیوتر، دانشگاه صنعتی اصفهان، اصفهان، montazeri@cc.iut.ac.ir
[3] استادیار دانشکده مهندسی کامپیوتر، دانشگاه اصفهان، اصفهان، bohlooli@eng.ui.ac.ir

چکیده –سیستم عامل میکروسی، یک سیستم عامل تعبیه‌شده متن باز است که در کاربردها و صنایع مختلفی از جمله صنعت هواپیمایی استفاده می‌شود. در این مقاله مکانیزمی جهت افزایش کارایی این سیستم عامل، از طریق مدیریت حافظه SPM، پیشنهاد شده است. SPM حافظه‌ای کوچک و سریع بوده که به صورت فیزیکی آدرس‌دهی شده درحالی‌که به فضای آدرس‌دهی مجازی نگاشت می‌شود. جهت بهره‌مندی از این حافظه، نیاز به مکانیزم‌های تخصیص داده‌ها به صورت موثر وجود دارد.الگوریتم پیشنهادی از سیاست صفحه‌بندی ساده، مکانیزم مدیریت حافظه سیستم عامل میکروسی و تفسیر برنامه‌نویس جهت تخصیص داده‌ها به این حافظه استفاده می‌کند. تمرکز اصلی این مقاله، مدیریت این حافظه در سطح سیستم عامل میکروسی است. نتایج شبیه‌سازی این مکانیزم نشان می‌دهد که 34.5% از محرومیت‌های ایجاد شده توسط برنامه‌ها جهت بهره‌مندی از این حافظه در الگوریتم مشابه بهبود یافته است.

کلید واژه–حافظه SPM، سیستم عامل میکروسی، سیستم‌های تعبیه‌شده، مدیریت حافظه پویا

1- مقدمه

سیستم‌های تعبیه‌شده[1]، سیستم‌های پردازش اطلاعاتی هستند که در محصول بزرگتر قرار می‌گیرند و از دید کاربر پنهان هستند؛ ازجمله این سیستم‌ها می‌توان به تجهیزات مخابراتی و سیستم‌های حمل و نقل اشاره کرد[1].

سیستم عامل میکروسی[2]، یک سیستم عامل تعبیه‌شده متن باز به زبان C است، که اهداف مدیریت زمان، مدیریت همزمانی وظیفه‌ها، مدیریت وقفه‌ها، سمافور و حافظه را در سیستم‌های تعبیه‌شده دنبال می‌کند[2]. این سیستم عامل از طرف اداره هوانوردی فدرال[3] مورد تایید قرار گرفته است و در دستگاه‌های خودکار هواپیما[4] کاربرد دارد[3].

واحد مدیریت حافظه این سیستم عامل به صورتی است که به جای استفاده از توابع malloc() و free()، از بلاک‌های حافظه با اندازه ثابت استفاده می‌کند. به این صورت که هنگام ایجاد حافظه پویا، این فضا را به صورت تعدادی بلاک با اندازه ثابت ایجاد کرده که هرکدام از آن بلاک‌ها به بلاک بعدی اشاره می‌کند؛ بنابراین یک لیست پیوندی یک‌طرفه از بلاک‌ها ایجاد می‌شود. به این لیست پیوندی، یک پارتیشن[5] گفته شده و توسط تابع OSMemCreate() ایجاد می‌شود. پس از آن با استفاده از تابع OSMemGet()، می‌توان هر یک از بلاک‌ها را از لیست پیوندی مربوطه خارج کرده و از آن بلاک استفاده کرد و درنهایت با اجرای تابع OSMemPut()، آن بلاک به لیست مربوطه برمی‌گردد[4].

SPM[6] حافظه‌ای کوچک و سریع بوده که نسبت به حافظه نهان درمصرف انرژی موثرتر عمل می‌کند[5] و به صورت فیزیکی آدرس‌دهی شده است درحالی‌که به فضای آدرس‌دهی مجازی نگاشت می‌شود؛ مولفه‌های سیستم عامل و کامپایلر از وجود چنین حافظه‌ای اطلاع دارند، بنابراین جهت استخراج مزیت‌های این حافظه، ارائه مکانیزم‌های موثر تخصیص حافظه توسط مولفه‌های فوق ضروری است[6].

یکی از معایب سیستم عامل میکروسی، عدم دسترسی به حافظه SPM می‌باشد؛ که روش ارائه شده در این مقاله با استفاده از مکانیزم مدیریت حافظه پویای این سیستم عامل، یک راهکار عملی جهت بهره‌مندی از حافظه SPM ارائه کرده است.

در بخش دوم برخی از کارهای انجام شده در جهت مدیریت حافظه SPM مطرح می‌شود. بخش سوم به معرفی روش پیشنهادی اختصاص داده شده است. در بخش چهارم نتایج پیاده‌سازی روش پیشنهادی ارائه می‌گردد. در نهایت در بخش پنجم خلاصه‌ای از آنچه در این مقاله بیان شده است ارائه می‌شود.

2- کارهای مربوطه

درسال‌های اخیر چندین روش جهت مدیریت حافظه SPM در سطح سیستم عامل بیان شده است که از جمله آن‌ها می‌توان به موارد زیر اشاره کرد.

درمرجع[7]، تکنیکی جهت تخصیص کدهای برنامه به حافظه SPM با هدف کاهش مصرف انرژی ارائه شده است؛ به‌این‌صورت‌که این حافظه به قسمت‌های مساوی تقسیم شده و هر قسمت به یک برنامه درزمان اجرا اختصاص می‌یابد. در [8]،دیدگاه تلفیق سخت‌افزار و نرم‌افزار تطبیق‌پذیر جهت مدیریت حافظه SPM در زمان اجرا با بالاسری پایین بر روی پردازنده مطرح شده است. هدف اصلی این مقاله بهبود توان در برنامه‌هایی است که از حافظه پویا استفاده می‌کنند. در این روش از یک واسط برنامه‌نویسی سطح بالا، جهت مدیریت این حافظه در زمان اجرا استفاده شده که توسط سیستم عامل مدیریت می‌شود.

درمرجع[9]، تکنیکی در سطح سیستم عامل جهت تخصیص داده‌های پویا به SPM مطرح شده است. بدین‌صورت که برنامه‌نویس را قادر می‌سازد تا کدهای برنامه را تفسیر کند. این تفسیر به صورتی انجام می‌شود که اگر برنامه‌نویس تشخیص دهد از داده‌های مورد نظر زیاد استفاده می‌شود، به آن‌ها اولویت بالا[۷] می‌دهد. بنابراین بهترین مکان ذخیره‌سازی این‌گونه داده‌ها، حافظه SPM خواهد بود؛ چرا که اگر در این حافظه قرار بگیرند، هزینه استفاده مجدد آن‌ها کاهش پیدا کرده و درنتیجه کارایی برنامه افزایش می‌یابد. درصورتی‌که برنامه‌نویس تشخیص دهد از داده‌های مورد نظر به طور مکرر استفاده نمی‌شود به آن‌ها اولویت پایین[۸] می‌دهد. بنابراین بهترین مکان ذخیره‌سازی این‌گونه داده‌ها، حافظه اصلی خواهد بود.

ازطرف دیگر، تخصیص داده‌ها توسط سیستم عامل به فضای قابل دسترس وابسته است؛ بنابراین سیستم عامل، تخصیص داده‌ها به حافظه SPM را ضمانت نمی‌کند[10]. به‌علاوه در این مرجع، امکان دارد درمحیط‌های چندوظیفه‌ای[۹] حداقل یکی از برنامه‌ها دچار محرومیت[۱۰] شود؛ چرا که هیچ نظارتی از طرف سیستم عامل (که وظیفه مدیریت حافظه را برعهده دارد) در جهت چگونگی تخصیص حافظه SPM به برنامه‌ها وجود ندارد.

3- الگوریتم پیشنهادی

هدف این مقاله این است که بتوان حافظه SPM را به‌صورتی در سطح سیستم عامل مدیریت کرد که داده‌هایی که در زمان اجرا ایجاد می‌شوند و بسیار مورد ارجاع قرار می‌گیرند را در این حافظه قرار داد به طوری که استفاده مجدد از آن‌ها با هزینه کمتر انجام گیرد و درنتیجه کارایی سیستم افزایش یابد. ازطرفی در یک محیط چند برنامه‌ای، تمام برنامه‌ها برحسب اولویتی که دارند بتوانند از این حافظه استفاده کنند.

برای حل این مسئله، از تلفیق روش ارائه شده در [9]، مدیریت حافظه پویای سیستم عامل میکروسی و سیاست صفحه‌بندی سیستم عامل استفاده شده است. به این طریق که در سه مرحله، زمان طراحی برنامه توسط برنامه‌نویس، زمان کامپایل توسط کامپایلر و زمان اجرا توسط سیستم عامل، حافظه SPM مدیریت می‌شود؛ در ادامه، این سه مرحله به تفصیل شرح داده می‌شود.

3-1- زمان طراحی برنامه‌ها

مطابق مکانیزم مدیریت حافظه پویای سیستم عامل میکروسی، برنامه‌نویس به‌جای استفاده از تابع ()malloc، حافظه مورد درخواستی خود را به صورت تعدادی بلاک با اندازه ثابت بیان می‌کند.

از طرف دیگر، مطابق روش ارائه شده در [9] برنامه‌نویس درخواست خود را تفسیر می‌کند. این تفسیر براساس میزان دسترسی به حافظه درخواست شده صورت می‌گیرد؛ به این صورت که اگر دسترسی به حافظه تقاضا شده زیاد باشد اولویت بالا و درغیر این صورت اولویت پایین داده می‌شود. بنابراین در زمان اجرا مشخص می‌شود که کدام داده‌ها در حافظه SPM می‌توانند قرار گیرند.

3-2- زمان کامپایل

قبل از زمان اجرا لازم است مراحل زیر توسط کامپایلر دنبال شود.

- قسمت مربوط به حافظه پویا SPM، به قسمت‌هایی با اندازه ثابت تقسیم می‌شود. هرکدام از این قسمت‌ها را یک قاب[۱۱] می‌نامیم. اندازه قاب برحسب تعداد برنامه‌های موجود، میانگین میزان حافظه پویا تقاضا شده برروی SPM و اندازه حافظه SPM تعیین می‌شود.

- کامپایلر به هرکدام از برنامه‌های موجود، اولویت پایین و یا بالا می‌دهد. تعیین اولویت برنامه‌ها برحسب ماهیت برنامه‌ها و مجموع میزان حافظه پویای تقاضا شده بر روی SPM است.
- به هر کدام از برنامه‌های موجود، برحسب اولویت آن‌ها یک تعداد قاب تعلق می‌گیرد. به طوری که تعداد قاب‌های تخصیص داده شده به برنامه‌های با اولویت بالا از برنامه‌های با اولویت پایین بیشتر خواهدبود؛ بنابراین هرکدام از برنامه‌های موجود سهمی از حافظه SPM خواهند داشت.

3-3- زمان اجرا

این مرحله نیز به دو زیرمرحله آماده‌سازی سیستم و اجرای برنامه‌ها تقسیم می‌شود.

3-3-1- مرحله آماده سازی

I. جهت مدیریت حافظه توسط سیستم عامل میکروسی، به ساختار بلاک کنترل قاب (FCB) به ازای هر قاب موجود درحافظه نیاز است که ساختار آن در شکل 1 نشان داده شده است.

II. در این مرحله سیستم عامل بر اساس تقسیم‌بندی‌های زمان کامپایل و با استفاده از FCBهای ایجاد شده دو لیست زیر را ایجاد می‌کند:

- لیست قاب‌های تخصیص داده شده: در ابتدا قاب‌هایی که به هر برنامه اختصاص داده شده است در این لیست قرار می‌گیرند. زمانی‌که برنامه درخواست حافظه پویا بر روی SPM بدهد و بخواهد از سهم خود استفاده کند قاب مربوطه از این لیست گرفته شده و پس از آنکه استفاده شد، قاب آزاد شده و به این لیست برمی‌گردد.
- لیست قاب‌های‌آزاد: با توجه به دقیق نبودن تقسیم‌بندی زمان کامپایل و اینکه سهم‌بندی حافظه SPM خطا خواهد داشت، بنابراین قاب‌های باقی‌مانده (قاب‌هایی که به هیچ‌کدام از برنامه‌ها تخصیص داده نشده است) در این لیست قرار می‌گیرد. اگر برنامه‌هایی بیش از سهم داده شده، حافظه بر روی SPM تقاضاکنند، سیستم عامل به آن درخواست‌ها از طریق این لیست پاسخ می‌دهد؛ و نیز پس ازآنکه از قابی که از این لیست برداشته می‌شود استفاده شد، (زمانی‌که برنامه آن قاب را آزاد کرد) آن قاب دوباره به این لیست باز می‌گردد.

```
typedef struct os_fcb{
INT8U     OSTaskId;
INT16U    OSFrameSize ;
INT16U    OSFreeSize;
INT8U     OSList;
INT8U     OSStatus;
void *    OSBeginAddr ;
void *    OSInternalAddr;
void *    OSNextFcb;
RECAP * OSRecapStrPtr;
FREE *    OSFreeStrPtr;
} OS_FCB;

typedef struct free {
void * next;
OS_MEM * part;
}FREE;

typedef struct recap {
void * next;
void * blk;
OS_MEM * part;
}RECAP;
```

شکل1: ساختار بلاک کنترل قاب

3-3-2-مرحله اجرای برنامه‌ها

همانطور که در شکل 2 نشان داده شده است، به ازای هربار درخواست حافظه بر روی SPM مراحل زیر دنبال می‌شود:

1) براساس اندازه درخواست مطرح شده، اندازه قاب و اندازه حافظه آزاد قاب‌هایی که در اختیار برنامه است مشخص می‌شود چند قاب جدید باید به این برنامه اضافه شود.
2) در صورتی‌که برنامه مورد نظر قابی در اختیار نداشته باشد، به سراغ لیست قاب‌های تخصیص داده شده رفته و سهم خود را برداشته و استفاده می‌کند. اگر سهم برنامه مورد نظر توسط برنامه دیگر درحال استفاده است، باید این سهم پس گرفته شده و در اختیار آن برنامه قرار داده شود که دراین صورت یک تاخیر قابل محاسبه خواهیم داشت.
3) در صورتی‌که برنامه مورد نظر تعدادی قاب در اختیار دارد، میزان فضایی که می‌توان از آن قاب‌ها به درخواست مربوطه داده شود، محاسبه می‌شود. به بیان دیگر، با توجه به اندازه بلاک‌ها و فضای آزاد قاب مشخص می‌شود چه تعداد بلاک از بلاک‌های مورد تقاضا در هر قاب قرار می‌گیرد.
4) در صورتی‌که نیاز به اخذ قاب جدید بود ابتدا بررسی می‌شود که آیا آن برنامه تعداد قاب‌های سهم خود را به طور کامل گرفته است یا خیر. در صورتی‌که از سهم خود استفاده نکرده است، مانند مرحله یک عمل می‌کند.

5) در صورتی‌که سهم خود را به طورکامل گرفته و نیاز به قاب جدید داریم، به سراغ لیست قاب‌های آزاد رفته و قاب را تخصیص می‌دهیم.
6) اگر درمرحله پنج، لیست قاب‌های آزاد خالی بود به سراغ سهم برنامه‌های دیگر رفته و درصورتی‌که برنامه‌ای دراین لحظه از سهم خود استفاده نکرده است (لیست قاب‌های تخصیص داده شده خالی نباشد) از سهم برنامه‌های دیگر استفاده می‌کنیم.
7) در صورتی‌که تا مرحله شش ادامه دهیم و هنوز فضای آزاد می‌خواهیم، باقیمانده فضای درخواستی کاربر را درون حافظه اصلی قرار می‌دهیم.
8) اکنون که قاب‌ها تخصیص داده شد، باید مطابق مکانیزم مدیریت حافظه سیستم عامل میکروسی، داخل قاب‌ها یک پارتیشن ایجاد کرده که شامل تعدادی بلاک با اندازه مساوی است. در انتها بلاک‌ها را به یکدیگر اشاره می‌دهیم.

باید توجه شود که چون لزوماً قاب‌ها پشت سر هم قرار نمی‌گیرند، با اعمال سیاست مدیریت حافظه سیستم عامل میکروسی و شکستن درخواست کاربر به تعدادی بلاک که به یکدیگر اشاره می‌کنند می‌توان با تخصیص دادن آن‌ها در قاب‌های مختلف، درخواست‌های حافظه با اندازه بالا را پاسخ داد. به بیان دیگر، بلاک‌های یک پارتیشن مورد درخواست برنامه را می‌توان در چهار گروه زیر قرار داد:

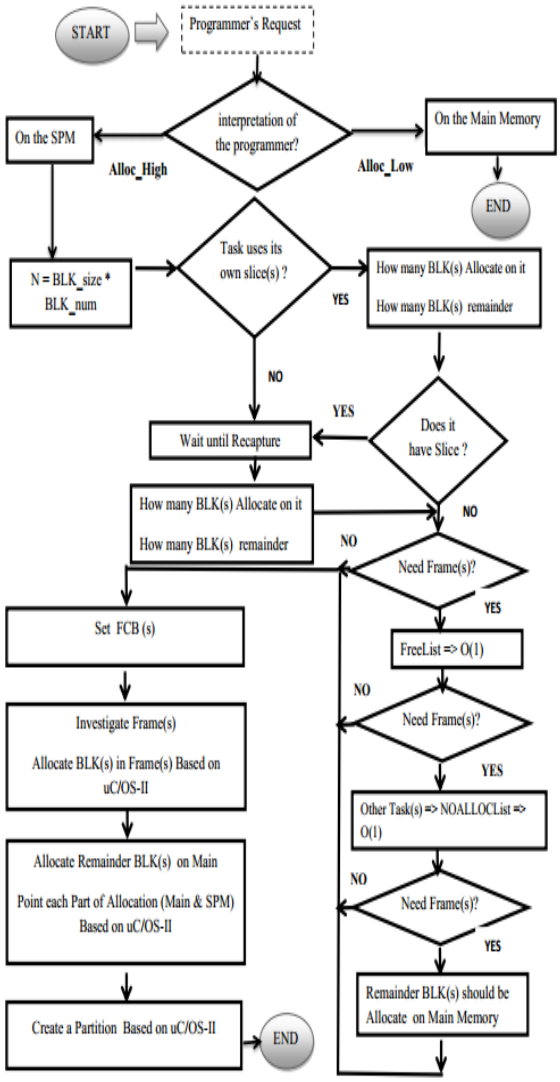
- بلاک‌هایی که در قاب‌های مربوط به سهم خود برنامه قرار می‌گیرند (Slice memory).
- بلاک‌هایی که در قاب‌های مربوط به لیست آزاد قرار می‌گیرند (FreeList memory).
- بلاک‌هایی که در قاب‌های مربوط به سهم برنامه‌های دیگر قرار می‌گیرند(Guest memory).
- بلاک‌هایی که نتوانستند در حافظه SPM تخصیص یابند، بنابراین به حافظه اصلی منتقل می‌شوند (Main memory).

فرمول (1)، محدوده اندازه بلاک‌های مورد درخواست را بیان می‌کند. باتوجه به اینکه بلاک‌ها به یکدیگر اشاره می‌کنند، اندازه آن‌ها نمی‌تواند کمتر از چهار بایت باشد و با توجه به اینکه بلاک‌ها باید درون قاب‌ها تخصیص یابند، اندازه آن‌ها نمی‌تواند از اندازه قاب بیشتر باشد. فرمول (2)، زمان انتظار مورد نیاز برنامه‌های درخواست‌دهنده را در بدترین حالت مشخص می‌کند. بدترین حالت زمانی است که کل قاب‌های مورد نظر حاوی بلاک‌های مورد درخواست برنامه‌های دیگر باشد. در این فرمول N، تعداد قاب‌های سهم برنامه مورد نظر است.

$$4(\text{Bytes}) \leq \text{Block Size} \leq \text{Frame Size(Bytes)} \qquad (1)$$

$$T_{Wait}(\text{WCET}) = N * \text{Frame Size} * (T_{SPM\ Read} + T_{Main\ Write}) \qquad (2)$$

زمانی‌که برنامه دستور حذف پارتیشن مربوط به درخواست حافظه SPM را صادر کرد، سیستم عامل ازطریق بلاک کنترل قاب، بلاک‌های آن پارتیشن را از قاب مربوطه حذف کرده و زمانی‌که تمام بلاک‌های موجود در یک قاب حذف شد، قاب مربوطه آزاد شده و به لیست متناظر خود بازمی‌گردد. این عمل با مرتبه زمانی O(1) انجام می‌گیرد؛ چراکه اگر فیلد OSFreeStrPtr موجود در بلاک کنترل آن قاب به Null اشاره کند، بدین مفهوم است که تمام بلاک‌های موجود در این قاب حذف شده‌اند.



شکل2: الگوریتم پیشنهادی

با توجه به اینکه اندازه قاب و تعداد قاب‌های سهم هر برنامه قبل از زمان اجرا مشخص می‌گردد و در طول اجرای برنامه‌ها ثابت در نظر گرفته می‌شود، بنابراین فرمول (2) که زمان انتظار مورد نیاز برنامه‌ها را در بدترین حالت در نظر می‌گیرد، تأثیر الگوریتم بیان شده را بر روی ویژگی بلادرنگ بودن سیستم عامل میکروسی مشخص می‌کند.

4- ارزیابی و شبیه‌سازی الگوریتم پیشنهادی

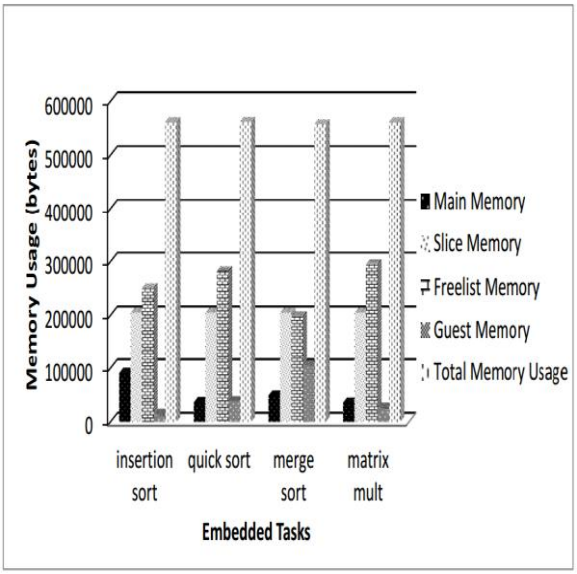
جهت ارزیابی روش پیشنهادی از نرم‌افزار Microsoft Visual Studio 2010 استفاده شد. برنامه‌های مرتب‌سازی درجی، مرتب‌سازی سریع، مرتب‌سازی ادغامی و ضرب ماتریس‌ها اصلاح[۱۲] شدند تا آن‌ها به صورت پویا حافظه‌های مورد نیاز خود را درخواست کنند. به علاوه به دو دلیل از این برنامه‌ها استفاده شده است. 1) این برنامه‌ها داده‌محور محسوب شده و الگوریتم پیشنهادی بر روی این نوع از برنامه‌ها طراحی شده است. 2) مرجع [11] این برنامه‌ها را به عنوان برنامه‌های تعبیه‌شده در نظر گرفته است.

جدول 1 پارامترهای شبیه‌سازی در نظر گرفته شده را بیان می‌کند. همانطور که در شکل 3 نشان داده شده است، بلاک‌های درخواستی برنامه در یکی از دسته‌های Main، Slice، Guest، Freelist قرار می‌گیرند. زمانی‌که هریک از برنامه‌ها درخواست حافظه بر روی SPM بدهند، به دلیل سهم‌بندی مطرح شده در الگوریتم، برنامه‌ها به خوبی از قاب‌های خود استفاده کرده و هیچ‌کدام از آن‌ها دچار محرومیت نمی‌شوند.

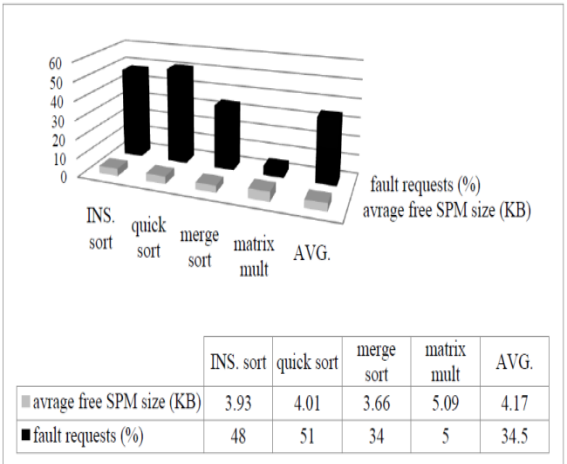
شکل 4 نشان می‌دهد که روش پیشنهادی در [9] با درنظر گرفتن پارامترهای مطرح شده در جدول 1، به طور میانگین 34.5% از درخواست‌های مربوط به حافظه SPM را بر روی حافظه اصلی منتقل می‌کند و این در حالی است که به طور میانگین 4KB حافظه آزاد بر روی SPM درآن لحظه وجود داشته که سیستم عامل نتوانسته است از آن حافظه استفاده کند. شکل 5 نشان می‌دهد که روش ارائه شده در این مقاله نسبت به روش مطرح شده در [9] نرخ برخورد[۱۳] بیشتری در شرایط یکسان به حافظه SPM داشته است چرا که از این حافظه بهینه‌تر استفاده کرده و می‌تواند داده‌های بیشتری را در این حافظه قرار دهد و نیز بیان می‌کند که این روش می‌تواند کارایی سیستم عامل میکروسی را بهبود دهد، چرا که افزایش نرخ برخورد به حافظه SPM باعث سریع‌تر شدن زمان پاسخ می‌شود.

جدول(1): پارامترهای شبیه‌سازی

Number of Tasks	4
SPM Size	16 KB
Main Memory Size	256 MB
Block Size for each Task	512 B
Frame Size	1024 B
Dynamic Memory Usage For Each Task	5 KB – 6 KB
Iteration For EachTask	100
Requests For SPM	400
Slices For Each Task	2 Frames

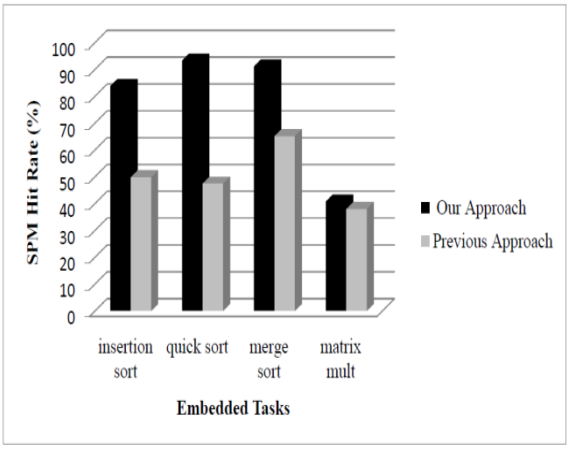


شکل3: نتیجه پیاده‌سازی SPM بر روی سیستم عامل میکروسی



	INS. sort	quick sort	merge sort	matrix mult	AVG.
avrage free SPM size (KB)	3.93	4.01	3.66	5.09	4.17
fault requests (%)	48	51	34	5	34.5

شکل4: شبیه‌سازی الگوریتم مطرح شده در [9]



شکل5: مقایسه نرخ برخورد

5- نتیجه‌گیری

سیستم عامل میکروسی یک سیستم عامل تعبیه‌شده و متن‌باز است که امروزه در صنایع نظامی ازجمله صنعت هواپیمایی مورد استفاده قرار می‌گیرد. در این تحقیق روشی مبتنی بر سیاست مدیریت حافظه پویای سیستم عامل میکروسی جهت مدیریت حافظه SPM در سطح سیستم عامل ارائه گردید. این مقاله از دو جهت قابل بررسی است : 1) در روش پیشنهادی، سیستم عامل از تفسیر برنامه‌نویس جهت تخصیص داده‌ها به حافظه SPM استفاده می‌کند ولی در عین حال با سهم‌بندی این حافظه، شکستن درخواست برنامه‌نویس به تعدادی بلاک با اندازه دلخواه و ثابت و تخصیص این بلاک‌ها در مکان‌های مختلف از محرومیت برنامه‌ها در محیط‌های چندبرنامه‌ای جهت بهره‌مندی از این حافظه جلوگیری می‌کند. 2) یک راهکار عملی در سطح سیستم عامل جهت بهره‌مندی از این حافظه ارائه شده است.

نتایج شبیه‌سازی نشان می‌دهد که هر کدام از برنامه‌ها به خوبی از قاب‌های خود جهت بهره‌مندی از حافظه SPM استفاده کرده و هیچ‌کدام از آن‌ها دچار محرومیت نشده‌اند؛ و نیز با مقایسه این روش با روش مشابه مشخص شده است که روش ارائه شده به علت شکستن درخواست برنامه‌نویس از حافظه SPM بهتر استفاده کرده است.

با توجه به اینکه کاهش توان مصرفی از دیگر ویژگی‌های حافظه SPM می‌باشد، برای ادامه این کار در آینده پیشنهاد می‌شود که تأثیر الگوریتم بیان شده در این مقاله بر روی توان مصرفی بررسی گردد.

مراجع

[1] Marwedel, P. (2003). Embedded system design (Vol. 1). Dortmund, Germany: Kluwer Academic Publishers.

[2] Jiawen, Zhang, Hong Liang, and Lu Xiaofeng. "An Improved Memory Management Method of uC/OS-II." Microwave Conference, 2008 China-Japan Joint. IEEE, 2008.

[3] http://www.micrium. com/about/story/

[4] Labrosse, J. (2002). The Real Time Kernel MicroC/OS-II.

[5] Takase, Hideki, Hiroyuki Tomiyama, and Hiroaki Takada. "Partitioning and allocation of scratch-pad memory for priority-based preemptive multi-task systems." *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2010*. IEEE, 2010.

[6] Yemliha, Taylan, et al. "SPM management using Markov chain based data access prediction." Computer-Aided Design, 2008. ICCAD 2008. IEEE/ACM International Conference on. IEEE, 2008.

[7] Verma, Manish, et al. "Scratchpad sharing strategies for multiprocess embedded systems: A first approach." *Embedded Systems for Real-Time Multimedia, 2005. 3rd Workshop on*. IEEE, 2005.

[8] Francesco, Poletti, et al."An integrated hardware/software approach for run-time scratchpad management." *Proceedings of the 41st annual Design Automation Conference*. ACM, 2004.

[9] Muck, Tiago Rogerio, and Antonio Augusto Frohlich. "Run-time scratch-pad memory management for embedded systems." IECON 2011-37th Annual Conference on IEEE Industrial Electronics Society. IEEE, 2011.

[10] Wasly, S., A Dynamic Scratchpad Memory Unit for Predictable Real-Time Embedded Systems,Electrical and Computer Engineering,Waterloo,Canada, 2012.

[11] Mittal, Shaily. "A New Efficient Replacement Policy for Scratch Pad Memory." Computational Intelligence and Communication Networks (CICN), 2013 5th International Conference on. IEEE, 2013.

زیرنویس‌ها

[1] Embedded System
[2] uC/OS-II
[3] Federal Aviation Administration
[4] Avionic
[5] Partition
[6] Scratch-pad Memory
[7] High
[8] Low
[9] Multi-Tasking
[10] Privation
[11] Frame
[12] Modify
[13] Hit Rate