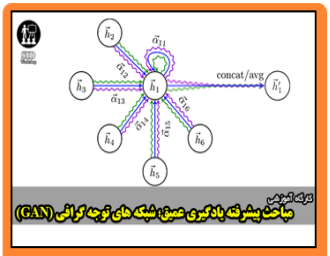
کارگاه های آموزشی مرکز اطلاعات علمی جهاد دانشگاهی

کارگاه آنلاین مکالمه روزمره انگلیسی

کارگاه آنلاین آموزش استفاده از وب آو ساینس



مباحث پیشرفته یادگیری عمیق؛
شبکه های توجه گرافی
(Graph Attention Networks)

انجمن کامپیوتر ایران
Computer Society of Iran

مرکز توسعه فناوری نیرو (متن)

ارائه‌ی یک مکانیزم جهت پورت کردن سیستم‌عامل‌های بی‌درنگ برای کاربردهای سیستم‌های نهفته

[1] محمد خادمی، [1] مقصود عباسپور، [2] سید امیر اصغری، [2] مرتضی انصاری‌نیا، [2] حسین پدرام

[1] دانشکده مهندسی برق و کامپیوتر، دانشگاه شهید بهشتی

[2] دانشکده‌ی مهندسی کامپیوتر و فناوری اطلاعات، دانشگاه صنعتی امیرکبیر

m.khademi@mail.sbu.ac.ir, maghsoud@sbu.ac.ir, {seyyed_asghari, m.ansarinia, pedram}@aut.ac.ir

چکیده: در کاربردهایی که نیاز به وظیفه‌های زیاد برای اجرا وجود دارد، می‌توان از یک سیستم‌عامل برای کنترل روند اجرای این وظیفه‌ها در سیستم‌های بی‌درنگ بهره برد. سیستم‌های بی‌درنگ، برای کاربرهایی در نظر گرفته می‌شوند که در آن‌ها وظایف بایستی در سرحدهای زمانی مشخص، انجام شوند. این دو پارامتر(زیاد بودن وظیفه‌ها و محدودیت‌های زمانی)، این التزام را ایجاد می‌نمایند که سیستم‌عامل مورد استفاده نیز بی‌درنگ باشد. پیرو این نیازها در این مقاله از هسته‌ی سیستم‌عامل µC/OS-II برای یک کاربرد نهفته بهره گرفته شده است. پورت کردن سیستم‌عامل‌ها بر روی بسترهای سخت‌افزاری همواره با این چالش روبروست که ابزار لازم وجود نداشته و اِشکال‌زدایی بر روی سیستم هدف بسیار پیچیده‌تر از سیستم‌های معمولی است. در این مقاله، علاوه بر ارائه‌ی یک مکانیزم برای پورت کردن سیستم‌عامل بی‌درنگ بر روی معماری مورد نظر، یک متد کلی برای پورت نمودن سیستم‌عامل‌های بی‌درنگ بر روی سیستم‌های نهفته‌ی جدید نیز ارائه می‌شود.

واژه های کلیدی: سیستم‌عامل، سیستم‌های بی‌درنگ، کاربرد نهفته، اِشکال‌زدایی

۱- مقدمه

سیستم‌های نهفته، کامپیوترهایی هستند که در دل بخش‌های محاسباتی سیستم‌های بزرگتر قرار گرفته و متکی بر پردازنده‌ی خود هستند. با توجه به پیشرفت‌های اخیر در در زمینه‌ی مجتمع سازی المان‌های الکترونیکی، کاربرد این سیستم‌ها، با سرعت بالایی در حال افزایش است. قطعات الکترونیکی خودروها، تلفن‌های همراه، چاپگرها و کاربری‌های خانگی همانند پخش‌کننده‌های صوتی-تصویری، اجاق‌های مایکروویو و غیره، نمونه‌هایی از این کاربردها هستند.

با آن‌که در نگاه اول، این کاربردها بسیار متفاوت از هم به نظر می‌آیند، اما همگی آن‌ها از یک ساختار کلی تبعیت می‌کنند. این ساختار، دریافت اطلاعات از ورودی‌ها(حسگرها)، پردازش آن‌ها(در پردازنده‌ها) و در نهایت انجام عکس‌العمل‌های مناسب(توسط عملگرها) است. همچنین می‌توان، قابلیت شبکه‌شدن را به عنوان یک ویژگی مشترک دیگر برای این سیستم‌ها عنوان نمود.

اغلب این سیستم‌ها به علت روند پیاده‌سازی سخت‌افزاری، قابلیت پیکربندی مجدد را ندارند(چرا که بصورت سیستم‌های بر روی تراشه پیاده‌سازی می‌گردند). این سیستم‌ها، معمولا برای یک کاربرد خاص طراحی می‌شوند و لذا دیگر نیازی برای درنظرگیری تمهیداتی برای پیکربندی مجدد وجود ندارد. لذا راه‌حل مناسب برای درنظرگیری تغییرات مورد نظر در این سیستم‌ها، تغیر زیرساختارهای نرم‌افزاری است.

راه‌حلی که برای جلوگیری از این پیچیدگی در سیستم‌های نهفته، پیشنهاد می‌شود، استفاده از یک سیستم‌عامل است. سیستم‌عامل‌ها، با ارائه‌ی یک سطح تجریدی از سخت‌افزار، برنامه‌نویسان را از درگیری با پیچیدگی‌های سخت‌افزاری مرتفع نموده، همچنین با عرضه‌نمودن سرویس‌هایی همانند فایل‌سیستم‌ها، درایورهای ورودی-خروجی و غیره، موجب تسهیل پیاده‌سازی نرم‌افزارهای کاربردی می‌شود. سیستم‌های عامل مسئولیت کنترل اجرای سیستم، زمان‌بندی وظیفه‌ها، مدیریت حافظه‌ی ورودی-خروجی، مدیریت فایل، اجرای همروند وظیفه‌ها را خود بر عهده می‌گیرند.

از آن‌جا که در سیستم‌های نهفته با محدودیت توان روبرو هستیم، لذا گزینه‌ی مناسب، بهره‌گیری از یک سیستم‌عامل نهفته‌ی بی‌درنگ است

که میزان خط کد کمی داشته و بتوان بـه راحتـی آن را اشـکال‌زدایـی نمود. علاوه بر این در سیستم‌های نهفتـه، قابلیـت حمـل بـه بسـترهای جدید بر عهده‌ی سیستم‌عامل نهفته است. یعنی این‌که نیازی به تغییـر برنامه‌های کاربردی برای اجرا بر روی بستر جدید نیست. در ایـن مقالـه جزئیات طراحی و پیاده سازی یک سیستم عامل نهفتـه‌ی بلادرنـگ بـا استفاده از هسته ای به نام µC/OS-II برای معمـاری ColdFire بررسـی می شود.

در بخش دوم مقاله، سیستم‌عامـل µC/OS-II معرفـی مـی‌گـردد. در بخش سوم، هسته‌ی مـورد اسـتفاده و بسـتر ColdFire معرفـی شـده و طراحی و جزئیات سیستم عامل توصـیف خواهـد شـد. در ایـن بخـش، همچنین رویه‌ی پورت هسته و ساخت سیسـتم عامـل براسـاس آن بـر روی بستر مورد نظر ارائه می شود. در بخش چهارم روند شروع بـه کـار سیستم‌عامل پس از پورت شدن بررسی می‌گردد. در نهایت بـه بررسـی روش‌های تست صحت انجام پورت و هم چنین پیاده سازی چند برنامه-ی کاربردی و نتایج تجربی به دست آمده می‌پردازیم.

۲- هسته‌ی سیستم‌عامل µC/OS-II

هسته‌ی سیستم‌عامل µC/OS-II به عنوان هسته‌ی یـک سیسـتم-عامل بی‌درنگ، دارای یک سری از ویژگی‌هـای شـاخص اسـت کـه بـه برخی از آن‌ها اشاره می‌شود[۱].

۲-۱ خاصیت قطعی‌بودن[1]

مدت زمان اجرایی تمامی توابع و سرویس‌ها قطعی است. ایـن بـدان معناست که می‌توانیم تعیین کنیم **µC/OS-II** چه مدت زمانی را برای اجرا شدن یک تابع و یا یک سرویس نیاز دارد. علاوه بر این بـه غیـر از یک سرویس، مدت زمان اجرای تمامی سـرویس‌هـای **µC/OS-II** بـه تعداد وظیفه‌های در حال اجرا بستگی ندارد.

۲-۲ سرویس‌ها

µC/OS-II تعـدادی از سـرویس هـای سیسـتمی را از جملـه **mailbox** ها، صف ها، سمافورها و **partition** های حافظه‌ای با سـایز ثابت و توابع وابسته به زمان و غیره را در اختیار می‌گذارد.

۲-۳ مدیریت وقفه‌ها

وقفه‌ها این قابلیت را دارند که اجرای یک وظیفه را متوقـف کـرده و اگر یک وظیفه‌ی با اولویت بالاتر بـه سیسـتم وارد شـود، وظیفـه‌ی بـا بالاترین اولویت را به محض اتمام وقفه های تودرتو می‌پذیرد. وقفـه‌هـا می‌توانند تا عمق ۲۵۵ سطح به صورت تودرتو اعمال شوند.

۲-۴ پشته‌های وظیفه‌ها

هر وظیفه، یک پشته‌ی مخصوص به خود را دارد. **µC/OS-II** به هر وظیفه این امکان را می‌دهد که پشته ای با سایز متفاوت داشـته باشـد. این خصوصیت ما را قادر می‌سازد تا میزان حافظه‌ی **RAM** مورد نیـاز را به حداقل برسانیم. همچنین این امکان داده می‌شود که تعیین کنیم هر وظیفه دقیقا چه میزان از پشته را نیاز دارد.

۲-۵ بخش بحرانی کد

بخش بحرانی کد که منطقه‌ی بحرانی نیـز نامیـده مـی‌شـود، قطعـه کدی است که نیاز دارد تا بگونه‌ای غیرقابل تقسیم و بصورت یک بلوک پایه‌ای[2] اجرا گردد. لذا قطعه کدی که در این بازه قرار می‌گیـرد، وقفـه نمی‌پذیرد. جهت این که از این موضوع اطمینان حاصـل شـود، تمـامی وقفه‌ها قبل از این‌که بخش بحرانی اجرا گردد، غیرفعال شـده و پـس از اجرای بحش بحرانی، دوباره به حالت قبل بازمی‌گردد.

۲-۶ قابلیت پورت

سیستم‌عامل µC/OS-II می‌توانـد بـه بسـیاری از ریزپردازنـده‌هـا و میکروکنترلرهای ۸، ۱۶، ۳۲ و ۶۴ بیتی پـورت شـود. نسـخه‌ی اولیـه‌ی µC/OS-II برای کاربردهای نهفته طراحـی شـد و ایـن مطلـب بـدین معنی است که اگر ابزار مناسبی(کامپایلر، اسمبلر و لینکـر) مناسـبی در اختیار باشد، می‌توان این سیستم‌عامل را بر روی محصول پورت نمود.

۲-۷ قابلیت گسترش

µC/OS-II بگونه‌ای خلاقانه طراحی شده است که تنها آن سرویس-هایی که در کاربرد نهفته به آن‌ها نیاز است، مـورد اسـتفاده قـرار مـی-گیرند؛ لذا کاربردی که سرویس‌های بیشتری نیاز دارد، به راحتـی مـی-تواند از آن‌ها بهره ببرد. قابلیت گسترش موجب مـی‌شـود تـا در میـزان استفاده از حافظه (RAM یا ROM) صرفه‌جویی زیادی شود.

۲-۸ قابلیت قبضه‌ای

µC/OS-II دارای یک هسته‌ی کاملا قبضه‌ای بی‌درنـگ اسـت. ایـن مطلب بدین معنی اسـت کـه µC/OS-II همـواره وظـایف بـا اولویـت بالاتری که آماده‌ی اجراست را اجرا می‌کند. بیشتر هسته‌های سیسـتم-عامل‌های تجاری[3]، قبضه‌ای هستند کـه µC/OS-II قابـل مقایسـه بـا بسیاری از آن‌هاست.

۳- رویه‌ی اجرا

در ادامه رویه‌ای که برای پورت کردن هسته‌ی سیستم عامل µC/OS-II انجام شده را به ترتیب انجام ارائه می‌کنیم. مراحل زیر نه تنها برای پردازنده‌ی ColdFire و سیستم عامل انتخابی صدق می-

[1] Deterministic

[2] Basic Block

[3] Commercial kernel

کنند، بلکه برای هر پردازنده و هر سیستم عامل بلادرنگ نهفته‌ای هم صادق هستند.

۳-۱ کتابخانه‌های مرتبط با معماری

علاوه بر کتابخانه‌هایی که به همراه کامپایلر برای کاربردهای عام تولید شده اند، نیاز به کتابخانه‌هایی برای کاربردهای سیستمی نیز داریم. به همین منظور از کتابخانه‌ی استاندارد C مواردی را انتخاب و برای کاربری سیستمی تغییر دادیم. از جمله‌ی این موارد می‌توان به stdlib، assert، و هم چنین تغییر alloc با توجه به معماری متفاوت و روش متفاوت سیستم عامل برای دسترسی به heap اشاره نمود. تعدادی از توابع دیگر نیز به خاطر روش‌های دسترسی به درایورها تغییر یافت. از جمله‌ی این موارد، توابع printf، scanf، getchar و ... می‌باشند که به صورت جزیی تر در ادامه بررسی می‌شوند.

همچنین به علت عدم ارائه کتابخانه‌هایی برای کاربردهای خاص این معماری، کتابخانه‌هایی برای دسترسی به عملیات غیراستاندارد مانند کنترل LEDها، دسترسی به پورت‌ها، دسترسی به حافظه‌های Flash (به صورت filesystem) و مدیریت وقفه‌ها ارائه شدند.

۳-۲ پورت قسمت‌های مختص به معماری

قابلیت پورت سیستم‌عامل، به صورت کلی در زمان طراحی هسته‌ی µC/OS-II در نظر گرفته شده بود. به همین منظور سیستم عامل به صورت لایه‌ای و در دو لایه‌ی مستقل از معماری و مرتبط با معماری طراحی شد. لایه‌ی مستقل از معماری به زبان C استاندارد، و لایه‌ی مرتبط با معماری به زبان ماشین نوشته می‌شود. سیستم باید در نهایت به‌گونه‌ای باشد که برنامه‌های کاربردی سیستم‌عامل مستقل از معماری باشند و نیازی به نگارش قسمتی از آن‌ها به کمک زبان ماشین نباشد.

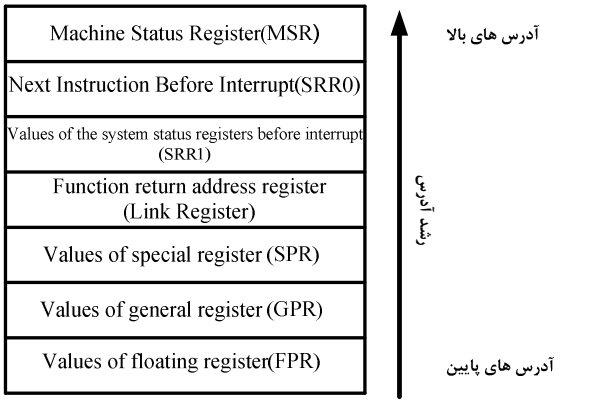
پردازنده‌ی هدف و کامپایلر در نظر گرفته شده باید دارای ویژگی‌های زیر باشند تا توانایی ساخت و اجرای هسته‌ی µC/OS-II را داشته باشند:

- توانایی اجرای توابع C به صورت reentrant در کامپایلر.
- پشتیبانی از وقفه در پردازنده و توانایی فعال‌سازی و غیرفعال سازی آن در کدهای C.
- توانایی انتقال همه‌ی داده‌های پشته به حافظه و بالعکس.
- تولید وقفه‌های زمانی توسط پردازنده.

با توجه به کمک گرفتن از کامپایلر gcc و ابزار مرتبط برای کامپایل سیستم‌عامل برای معماری ColdFire و همچنین مشخصات پردازنده مورد استفاده، تمامی موارد فوق مورد پشتیبانی می‌باشند.

هسته‌ی µC/OS-II به گونه‌ای پیاده‌سازی شده است که چند تابع مهم سیستم‌عامل باید به کمک زبان ماشین پیاده سازی شوند. این توابع که نقش تعیین‌کننده‌ای در اجرای درست سیستم دارند، از قرار زیر هستند:

- **تابع OSTaskStkInit** که برای مقداردهی اولیه پشته‌ی هر وظیفه به کار می‌رود. این تابع باید با توجه مدل پشته‌ای پردازنده‌های ColdFire پیاده سازی شود. مدل پشته‌ای این پردازنده‌ها، در شکل ۱ نشان داده شده است[۴].



شکل ۱ مدل‌پشته‌ای پردازنده‌های ColdFire

- **توابع OSCtxSw و OSIntCtxSw** که برای تغییر متن در سیستم‌عامل در هنگام یک رخداد مناسب یا در هنگام پایان یافتن بازه‌ی زمانی اجرا می شوند. این توابع، اجرای بهترین وظیفه‌ی قابل اجرا را برعهده دارند.

- **تابع OSStartHighRdy** که باید بهترین وظیفه‌ی آماده‌ی اجرا را در زمان شروع به کار سیستم‌عامل، انتخاب و اجرا کند.

- **تابع TickISR** که اغلب روتین سرویس وقفه‌ی زمانی است و در هر بار رخ دادن این وقفه، در صورت امکان، عمل تعویض متن[۴] را انجام می‌دهد.

- **تابع OS_ENTER_CRITICAL و تابع OS_EXIT_CRITICAL** که به ترتیب برای نشان دادن شروع و پایان ناحیه بحرانی به کار می‌روند. این توابع با توجه به کاربرد خاص شان باید با دقت در کدهای کاربر استفاده شوند. استفاده از این توابع در کدهای کاربر می‌تواند با استفاده از سمافور، تعویض شود. کاربری اصلی این توابع بیشتر در هسته و درایورها است. با توجه به این موارد، این دو تابع بایستی به صورت حداقلی طراحی شوند تا تاثیر زیادی بر سرعت سیستم‌عامل نگذارند. پیاده‌سازی این دو تابع، برای عدم تداخل در کار سیستم به صورت شبه کد شکل ۲ انجام می‌پذیرد:

[4] Context Switch

```
OS_ENTER_CRITICAL{
        Save Status Register Value into currently running task stack
        Disable interrupts
}
OS_EXIT_CRITICAL{
        Restore top of current task stack into Status Register
}
```

شکل ۲ پیاده‌سازی توابع OS_Enter_Critical و OS_Exit_Critical

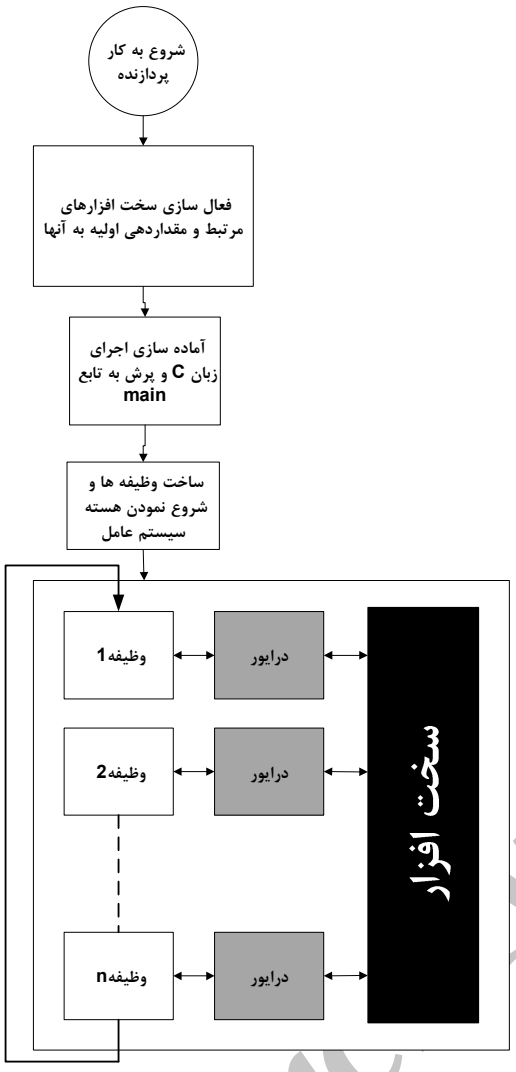
علاوه بر موارد فوق، برای یکسان سازی تعریف متغیرها، باید اندازه‌ی متغیرهای مختلف و همچنین جهت افزایش پشته را با توجه به کامپایلر و معماری پردازنده مشخص کنیم. این عمل در فایل os_cpu.h که از فایل های مرتبط با معماری در سیستم‌عامل می‌باشد، انجام می‌گیرد. با توجه به اطلاعات پردازنده[۲] [۳]، این مقادیر به شرح زیر است:

جدول ۲- اندازه ی متغیرها در معماری Coldfire ویرایش 4e

نوع متغییر	اندازه (بیت)
int	۳۲
Char	۸
Short	۱۶
Float	۳۲
Double	۶۴

۴- شروع به کار سیستم‌عامل

همان طور که اشاره شد، µC/OS-II تنها شامل هسته‌ی سیستم عامل است. برای دستیابی به یک سیستم عامل کامل، علاوه بر هسته نیاز به Bootloader، درایورهای مناسب، روتین های وقفه و هم چنین وظیفه های مورد نظر داریم. شکل ۳ مسیر روشن شدن پردازنده تا شروع به کار وظیفه‌ها به عنوان هدف را نمایش می دهد.



شکل ۳ دیاگرام اجرای سیستم‌عامل از زمان روشن شدن سیستم تا شروع اجرای وظیفه‌ها

همانطور که در شکل۳ مشاهده می‌شود، در ابتدا نیاز به مقدارگذاری-های اولیه و آماده به کار نمودن بخش‌های مختلف سخت افزاری و نرم افزاری داریم. به بیان دیگر باید سخت‌افزار را برای اجرای هسته‌ی سیستم‌عامل و وظیفه‌های طراحی شده، آماده کنیم. ترتیب این عملیات به شرح زیر است:

۱. مقداردهی اولیه و آماده‌سازی رجیسترهای کنترلی، بردار وقفه، حافظه، پشته و heap برای اجرای کُدهای زبان C.

۲. پرش به تابع main اصلی در زبان C و اجرای آن.

۳. آماده سازی هسته سیستم عامل.

۴. ساخت وظیفه‌های طراحی شده با توجه به ساختار مورد انتظار و

افزودن آنها به صف وظیفه های آماده به کار.

۵. آغاز نمودن سیستم عامل که شامل انتخاب وظیفه‌ی آماده با اولویت بالاتر است و اجرای آن.

۶. تغییر وظیفه‌ی در حال اجرا، در صورت پایان یافتن وظیفه‌ی در حال اجرای فعلی، یا وجود وظیفه‌ای آماده اجرا با اولویت بالاتر در زمان وقوع وقفه‌ی زمانی و زمان بندی مجدد.

موارد ۱ و ۲ توسط نرم افزار جداگانه‌ای به نام Bootloader انجام می‌شود که مستقل از سیستم عامل است و تنها در هنگام شروع به کار پردازنده فراخوانی می‌شود. با شروع به کار هسته‌ی سیستم‌عامل، کنترل پردازنده به دست هسته‌ی سیستم عامل می‌افتد و تا هنگام راه‌اندازی مجدد سیستم(و یا بروز مشکل حاد) این اختیار در دست هسته می‌ماند.

در هنگام تغییر متن توسط هسته در مورد ۶ از لیست ارائه‌شده، تمامی اطلاعات وظیفه‌ی در حال اجرا، اعم از مقادیر فعلی رجیسترها عمومی داده و آدرس، مقدار رجیستر وضعیت، مقادیر رجیسترهای واحد محاسبات نقطه شناور و آدرس دستورالعملی که وظیفه در آن متوقف شده است بر روی پشته‌ی همان وظیفه ذخیره می‌شود. در هنگام اجرای مجدد این وظیفه، این اطلاعات از روی پشته برداشته شده و به مکان‌های مناسب باز می‌گردند. شبه الگوریتم شکل ۴، عمل تعویض متن در تابع OSCtxSw و OSIntCtxSw را نشان می‌دهد.

```
Context Switch{
    save all registers into the task stack
    save current TCB into the task stack
    select best task to run based  on scheduling result
    restore this task TCB, registers, etc.
    Change currently running TCB
    go to the actvie task codes
}
```

شکل ۴ شبه الگوریتم عمل تعویض متن در تابع OSCtxSw و OSIntCtxSw

۵- درایورها

برای ارتباط نرم افزار با سخت افزارهای موجود بر روی پردازنده یا در ارتباط با آن (بر روی بورد) نیاز به درایورهای واسطی است که کاربر به کمک آن‌ها و بدون دغدغه‌ی ریزه‌کاری‌های سخت افزاری، از سخت افزار استفاده کند. به همین منظور درایور بعضی از سخت‌افزارهای پرکاربرد مانند پورت سریال، چراغ‌های LED و پورت CAN پیاده سازی شد. به کمک این درایورها، برنامه نویس می‌تواند با کُدهای سطح بالا در زبان C به سخت افزار دسترسی داشته باشد.

برای دسترسی به پورت سریال، تابع printf از کتابخانه‌ی استاندارد stdio طوری تغییر یافت که به جای چاپ بر روی صفحه نمایش، داده‌ها را به پورت سریال ارسال نماید. برنامه نویس تنها با ارسال یک رشته به تابع printf آن رشته را به خروجی پورت سریال ارسال می‌کند. برای نمونه فراخوانی دستور زیر، مقدار عددی متغییر temprature را از طریق پورت سریال به خارج از پردازنده ارسال می نماید.

```
printf (“temprature is %i”,temprature);
```

برای دریافت اطلاعات از پورت سریال نیز، به کمک روتین وقفه‌ی دریافت در پورت سریال، داده‌های دریافتی، در بافری که به همین منظور در سیستم عامل تعبیه شده، قرار می‌گیرد. وظیفه‌هایی به داده‌های دریافتی از پورت سریال نیاز دارند، به وسیله‌ی دستورات استاندارد scanf و getchar می‌توانند داده‌ها را دریافت کنند. درایور پورت CAN نیز به همین صورت با توابع printf و scanf پیاده سازی شده است.

نکته‌ی مهم در طراحی و پیاده‌سازی درایورها، تفاوت و پیچیدگی بیشتر آن‌ها در سیستم‌عامل، در مقابل نرم‌افزارهای غیرسیستمی است. در صورتی که چند وظیفه بخواهند همزمان به سخت افزار دسترسی داشته باشند، باید از سمافور یا ناحیه‌ی بحرانی برای دسترسی به سخت افزار استفاده نمود. برای نمونه، در هنگام فراخوانی تابع ارسال به پورت سریال یا CAN، سیستم عامل وارد ناحیه بحرانی می‌شود و از تعویض متن یا بروز وقفه‌ی تودرتو جلوگیر می‌کند تا ارسال داده دچار مشکل نشود.

چراغ های LED نیز به وسیله ی توابع led_on، led_off و led_toggle کنترل می‌شوند. این توابع که در کُدهای هر وظیفه قابل دسترسی است، به ترتیب موجب روشن شدن، خاموش شدن و تغییر وضعیت LED های متصل به بورد می‌شوند. انتخاب LED نیز به وسیله‌ی پاس کردن شماره‌ی آن به این توابع صورت می‌گیرد.

۶- وقفه‌ها

برای ارتباط سخت افزار با نرم افزار و اطلاع سیستم عامل و وظیفه ها از رخدادهای سخت‌افزاری از وقفه استفاده می‌شود. هر وقفه در صورت بروز، یک تابع مرتبط با آن که در بردار وقفه تعیین می‌شود را فراخوانی می‌کند. این جدول بردار در هنگام شروع به کار پردازنده توسط Bootloader در حافظه پردازنده قرار می‌گیرد. شکل ۵ بخشی از این جدول را که برای اعمال تعویض متن در هنگام بروز وقفه‌ی زمان‌سنج انجام می‌شود را نشان می‌دهد.

```
vector28:   .long   asm_exception_handler   /* TRAP #8
vector29:   .long   asm_exception_handler   /* TRAP #9
vector2A:   .long   asm_exception_handler   /* TRAP #10
vector2B:   .long   asm_exception_handler   /* TRAP #11
vector2C:   .long   asm_exception_handler   /* TRAP #12
vector2D:   .long   asm_exception_handler   /* TRAP #13
vector2E:   .long   OSIntCtxSw              /* TRAP #14
vector2F:   .long   asm_exception_handler   /* TRAP #15
vector30:   .long   asm_exception_handler   /* Reserved
```

شکل ۵ قسمتی از جدول بردار وقفه

۷- اِشکال‌زدایی

اشکال‌زدایی از نرم‌افزارهای سیستم‌های نهفته مشکلات مخصوص به خود را دارد. این کار در سیستم‌های نهفته متفاوت و سخت‌تر از نرم-افزارهای معمولی است. در کنار این، سیستم عامل‌ها نیز علاوه بر اینکه یکی از نقاط پر اِشکال در طراحی و پیاده سازی می‌باشند، از روش متفاوت و پیچیده‌ی اشکال‌زدایی بهره می‌برند. چرا که ابزاری که برای اشکال‌زدایی عادی به کار می‌روند، خود نیاز به سیستم عامل جهت اجرا دارند. به همین خاطر باید به صورت remote سیستم عامل را اشکال زدایی نمود.

اشکال‌زدایی سیستم طراحی‌شده در دو مرحله انجام می‌شود:

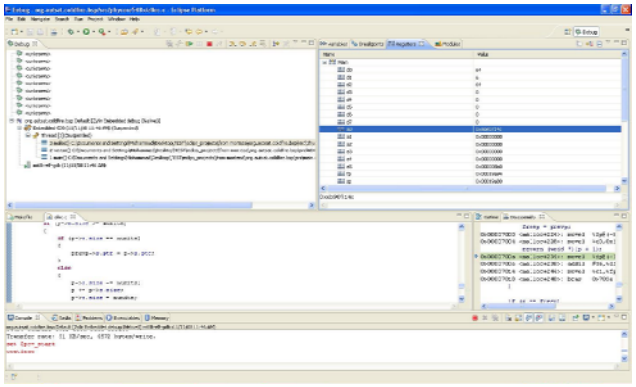
۱. اِشکال‌زدایی بر روی شبیه‌ساز سخت‌افزار

۲. اِشکال‌زدایی بر روی سخت‌افزار واقعی

در مرحله‌ی اول، اشکالات منطقی سیستم عامل آشکار و رفع گردید. برای شبیه‌سازی سخت افزار از شبیه‌سازهای ColdFire Emulator و SkyEye بهره گرفته شد.

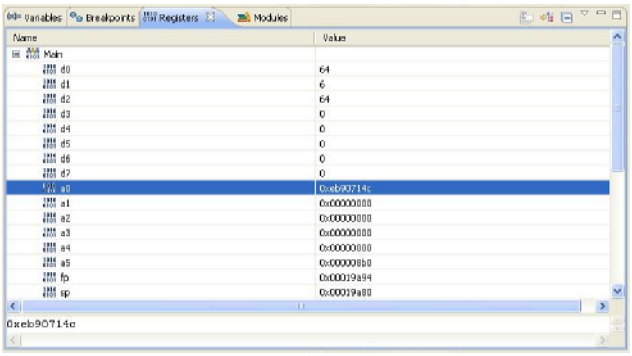
در مرحله‌ی دوم، اشکال زدایی بر روی سخت افزار واقعی و به کمک واسط اشکال‌زدایی BDM انجام می‌گردد. در این مرحله، اشکالات درایورها و عدم سازگاری‌های سخت افزار با نرم افزار رفع می‌شود. به علت رفع اشکالات منطقی در این مرحله، تنها اشکالاتی که نیاز به توجه سخت افزاری دارند، رخ می‌دهند.

برای اشکال‌زدایی توسط واسط BDM، نرم‌افزارهای کمکی جهت ارتباط با این واسط به کمک gdb تحت محیط توسعه‌ی نرم‌افزار Eclipse ساخته شد. در شکل ۶ محیط اشکال زدایی ساخته‌شده را مشاهده می‌کنید.



شکل ۶ محیط توسعه‌ی اِشکال‌زدایی Eclipse

این محیط، توانایی نمایش تابع فعلی در حال اجرا، مقادیر فعلی رجیسترها و در کل پارامترهای مختلف پردازنده و مقادیر متغیرهای سیستم عامل را دارد. در شکل ۷ طرز نمایش مقادیر رجیسترهای پردازنده را مشاهده می‌نمایید.



شکل ۷ نمایش مقدار رجیسترهای پردازنده

۸- نتایج تجربی

برای بررسی قابلیت کارایی این سیستم عامل در مقابل وظیفه‌هایی که بر روی آن اجرا می‌شود، سه برنامه‌ی کاری مورد استفاده قرار گرفت:

۱. برنامه‌ی کاری مرتب‌کردن اعداد(در این رویه، از مرتب‌سازی حبابی بهره گرفته شد)

۲. برنامه‌ی کاری ضرب دو ماتریس ۴*۴

۳. برنامه‌ی کاری تقریب عدد پی.

این برنامه‌های کاری با استفاده از زبان C نوشته شدند.

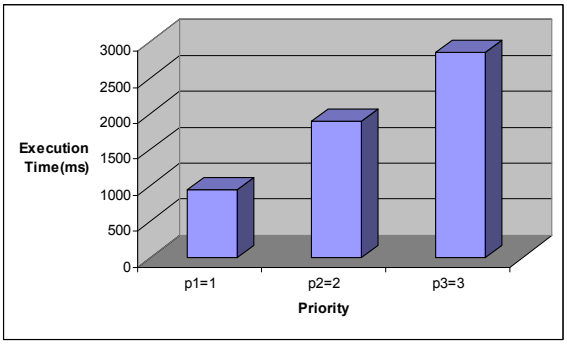
آزمایشات مورد نظر برای رسیدن به دو هدف انجام شد:

۱. تایید همروندی اجرا و انجام وظیفه‌ها بر روی سیستم‌عامل بی‌درنگ

۲. تایید تاثیرپذیری سیستم‌عامل از اولویت‌های انتساب‌شده به وظیفه‌ها بصورت بی‌درنگ

برای رسیدن به هدف نخست، سه برنامه‌ی کاری ذکر شده، به تعداد ۳۰۰۰ بار(هر کدام از وظایف ۱۰۰۰بار) اجرا شدند، که این اجرا بصورت همروند انجام شد.

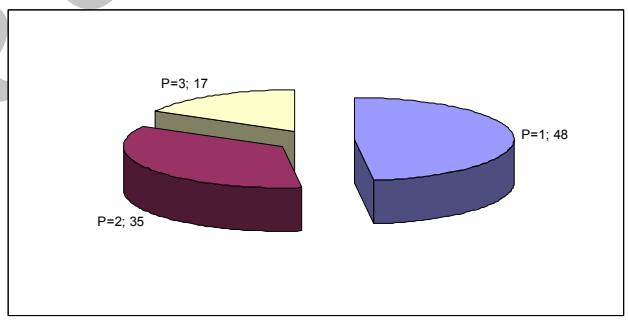
برای رسیدن به هدف دوم، یعنی تاثیرپذیری سیستم‌عامل از اولویت-های انتساب‌شده، برنامه‌ی کاری مرتب‌سازی حبابی به عنوان یک وظیفه در نظر گرفته شد. حال به این وظیفه، سه اولویت مختلف انتساب گردید(اولویت‌های ۱، ۲و ۳). در سیستم‌عامل µC/OS-II وظیفه‌های با اولویت عددی پایین‌تر دارای اولویت اجرای بالاتری هستند. لذا همانطور که در شکل۸ مشاهده می‌کنید و همان‌طور که مورد انتظار نیز بود، وظیفه‌ی با اولویت ۱ سریعتر از وظیفه‌ی با اولویت ۲ و اولویت با وظیفه‌ی ۲ نیز سریعتر از وظیفه‌ی با اولویت ۳ اجرا گردید.



شکل ۸ نسبت زمان اجرا، متناسب با اولویت‌های انتساب‌شده به وظیفه‌ها

در این آزمایش، به وظیفه‌ی مورد نظر، سه اولویت ۱، ۲ و ۳ انتساب شد. وظیفه‌ی با اولویت ۱ پس ۹۳۷ میلی‌ثانیه، وظیفه‌ی با اولویت ۲، پس از ۱۸۹۱ میلی‌ثانیه و وظیفه‌ی با اولویت ۳ پس از ۲۸۵۱ میلی‌ثانیه به پایان رسید که به زمان‌های خاتمه‌ی ایده‌آل بسیار نزدیک است.

در آزمایش بعدی پس از ۲ ثانیه از اجرای همروند سه وظیفه‌ی همسان با اولویت‌های ۱، ۲ و ۳، اجرا را قطع کرده و میزان سهم تخصیص پردازنده را به هر یک از وظیفه‌ها تحلیل نمودیم که در شکل ۹ نمایش داده شده است.



شکل ۹ سهم در اختیارگیری پردازنده برای وظیفه‌های با الویت‌های ۱، ۲ و ۳

لذا همان‌گونه که مشاهده می‌شود و مورد انتظار نیز هست، نسبت تخصیص پردازنده به وظیفه‌ی با اولویت ۱، تقریبا ۳ برابر وظیفه‌ی با اولویت ۳ و ۱/۵ برابر وظیفه‌ی با اولویت ۲ است.

۹- نتیجه گیری

در این مقاله، پورت کردن یک سیستم‌عامل بی‌درنگ، مستقل از نوع سیستم‌نهفته، مورد بررسی قرار گرفت. روش ارائه‌شده، برای سیستم‌های بی‌درنگ دیگر نیز قابل تعمیم است.

محصول روش ارائه‌شده، سیستم‌عاملی کامل مشتمل بر ابزار بوت، هسته‌ی سیستم‌عامل، سیستم‌ارتباط وظیفه‌ها و همچنین درایورهای سخت‌افزار است. به کمک ابزار ارائه‌شده، برنامه‌نویس دیگر نیاز به توجه به جزئیات سخت‌افزاری ندارد.

علاوه‌بر سیستم‌عامل نهفته‌ی ارائه‌شده، ابزاری جهت اِشکال‌زدایی و همچنین تست سیستم‌عامل و وظیفه‌ها در محیط شبیه‌سازی شده و درنهایت محیط واقعی ارائه‌شده است. به کمک این ابزار، هزینه‌ی توسعه‌ی سیستم‌عامل جهت یک بستر سخت‌افزاری نهفته کاهش می‌یابد.

مراجع

[1] Jean J. Labrosse, MicroC/OS-II, The Real-Time Kernel, 2nd edition CMP Books, 2002.

[2] MCF548x Reference Manual Devices Supported: MCF5485 MCF5482, MCF5484 MCF5481, MCF5483 MCF5480, Document Number: MCF5485RM, Rev. 4, 07/2006

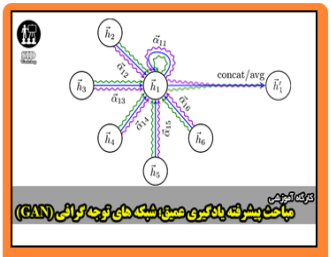
[3] ColdFire®CF4e Core User's Manual, Freescale Semiconductor Inc, V4ECFUM/D Rev. 0, 06/2001

[4] LIU Kai , XU Hong-zhe , XIA Wei-ran , HE Hong-bo, Analysis and Implementation of Porting µCOS-II to MPC555, Journal of Communication and Computer, Sep. 2005, Volume 2, No.9

کارگاه های آموزشی مرکز اطلاعات علمی جهاد دانشگاهی

کارگاه آنلاین مکالمه روزمره انگلیسی

کارگاه آنلاین آموزش استفاده از وب آو ساینس



مباحث پیشرفته یادگیری عمیق؛ شبکه های توجه گرافی (Graph Attention Networks)