**گزارش دستورکار هشتم آزمایشگاه سیستم­های عامل، قسمت دوم**

نگار موقتیان، 9831062

**بهبود الگوریتم FCFS**

الگوریتم FCFS ساده­ترین نوع الگوریتم­های زمانبندی بوده و به طور کلی برای زمانی مناسب است که پردازه­هایی با waiting time کوتاه و نسبتاً یکنواخت داریم. اگر طول پردازه­ها بسیار متفاوت باشد این خطر وجود دارد که پردازه­های کوتاه مجبور شوند برای اجرا منتظر پردازه­های طولانی مانده و در نتیجه اثر کاروان به وجود آمده و متوسط زمان اجرای برنامه­ها زیاد شود. همچنین این الگوریتم ذاتاً با اجرای موازی پردازه­ها و پایپ­لاین­ سازگار نیست.

بنابراین برای بهبود این الگوریتم نیاز است آن را به صورتی تغییر دهیم که پردازه­های طولانی نتوانند تا اتمام کار خود CPU را بدست بگیرند و پردازه­های کوچک­تر مجبور نباشند تا انتها منتظر آن­ها بمانند. اگر این الگوریتم را به گونه­ای تغییر دهیم که پردازه­های کوچک­تر در ابتدا اجرا شوند این الگوریتم حالت SJF به خود می­گیرد و اگر آن را به گونه­ای تغییر دهیم که به صورت قبضه­ای عمل کند و نگذارد پردازه­های طولانی­تر بیش از یک حد مشخص اجرا شوند الگوریتم حالت RR به خود می­گیرد.

می­توان از تکنیک­های دیگری نیز برای بهبود الگوریتم استفاده کرد. برای مثال می­توانیم به این صورت عمل کنیم که اگر زمان اجرای یک پردازه پردازه­ای کوتاه­تر وارد سیستم شد شانس اجرا را به او داده و سپس به پردازۀ قبلی بازگردیم و در غیر این صورت به اجرای پردازه­ها به صورت FCFS ادامه دهیم.

یا تا مدتی مشخص به پردازه زمان اجرا داده و پس از آن پردازه را قبضه کرده، پردازنده را در اختیار پردازۀ بعدی موجود در صف FCFS قرار دهیم و پس از سپری شدن مدت مشخص برای پردازۀ دوم باز به پردازۀ اول باز گردیم. در این صورت پردازه­های متنوع­تری فرصت اجرا پیدا خواهند کرد و احتمال اجرای پردازه­های کوچک­تر افزایش می­یابد.

**بهبود الگوریتم SJF**

برای الگوریتم SJF اثبات می­شود که می­توان کمترین میانگین زمان انتظار را داشت و این الگوریتم از این نظر به خوبی عمل می­کند. اما مشکل این الگوریتم این است که با این روش ممکن است پردازه­های طولانی دچار قحطی شده و پردازنده به آن­ها اختصاص نیابد.

می­دانیم الگوریتم SJF را می­توان گونه­ای از الگوریتم مبتنی بر الویت در نظر گرفت که در آن الویت پردازه­ها متناظر با زمان اجرای آن­هاست (و لذا پردازه­های طولانی الویت کم­تری خواهند داشت). بنابراین برای اینکه از قحطی در این الگوریتم جلوگیری کنیم می­توانیم از aging در پردازه­ها استفاده کنیم. برای این کار در ابتدای ورود هر پردازه به عنوان الویت زمان burst time آن را تخصیص می­دهیم و در طول زمان از این عدد الویت کم می­کنیم تا الویت پردازه افزایش یابد. از این طریق پردازه­های کوتاه­تر در الویت قرار خواهند گرفت اما پردازه­های طولانی نیز به مرور زمان شانس اجرا شدن پیدا خواهند کرد و لذا دچار قحطی نخواهند شد.

**بهبود الگوریتم مبتنی بر الویت**

الگوریتم مبتنی بر الویت بر خلاف الگوریتم­های دیگر الویت پردازه­ها را نیز در نظر می­گیرد و از این جهت برای استفاده در سیستم­های معمول (که در آن­ها کارها الویت­های متفاوتی دارند) برتری دارد. اما این روش نیز خطر قحطی برای پردازه­های کم الویت را در پی دارد.

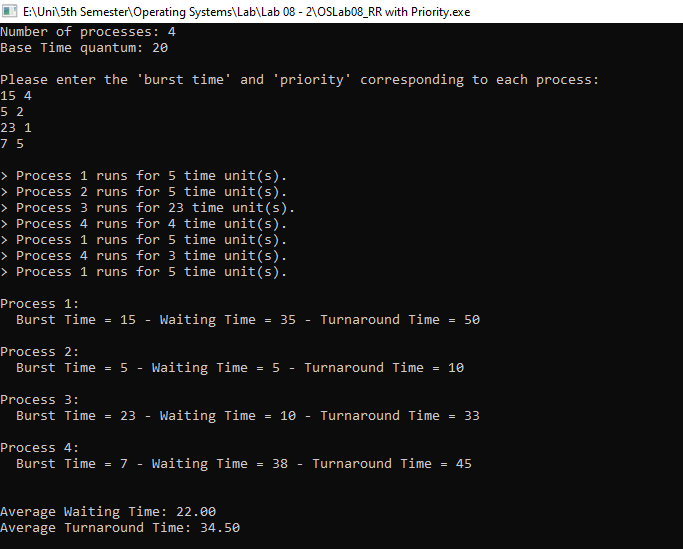
همانطور که در قسمت قبل نیز اشاره شد برای جلوگیری از قحطی پردازه­های کم الویت می­توانیم از aging استفاده کنیم. با استفاده از این تکنیک در هر مرحله به الویت پردازه­ها اضافه خواهد شد و به طور طبیعی هر پردازه­ای که قدیمی­تر باشد به مرور الویت بیش­تری کسب خواهد کرد و لذا به مرور شانس اجرا شدن آن افزایش یافته و از قحطی آن جلوگیری می­شود.

**بهبود الگوریتم RR**

الگوریتم RR از این جهت اهمیت دارد که با چرخش نوبت میان پردازه­ها تضمین می­کند هیچ یک از آن­ها دچار قحطی نخواهند شد. در عوض عملکرد این الگوریتم (از نظر میانگین زمان انتظار و اجرای پردازه­ها) ممکن است به خوبی الگوریتم­هایی مانند SJF نباشد. به علاوه راه حلی برای پردازه­های با الویت بالا (و علی­الخصوص NMI ها) که باید پیش از پردازه­های دیگر و به صورت یک­تکه اجرا شوند ارائه نمی­دهد و به عبارتی تمام پردازه­ها را به چشم یکسان می­بیند.

برای بهبود این مشکل پردازه­هایی که می­خواهند به شکل RR اجرا شوند باید یک الویت نیز داشته باشند. پس از آن اینکه چه time quantum ای برای اجرا به آن­ها اختصاص داده شود و یا اساساً تکه تکه شوند یا خیر می­تواند بر اساس این الویت تصمیم­گیری شود. برای مثال می­توان به پردازه­هایی با الویت بالاتر زمان بیش­تری برای هر دور اجرا اختصاص داد (می­توان یک time quantum اولیه داشت و این مقدار بر الویت پردازه تقسیم شود. از این طریق پردازه­های با شماره الویت کمتر و الویت بیش­تر زمان بیش­تری برای اجرا خواهند داشت). به علاوه می­توان تعیین کرد که اگر الویت پردازه از مقدار خاصی بیش­تر بود اصلاً آن را تکه­تکه نکنیم و اجازۀ اجرا تا به انتها را به آن بدهیم.

پیاده­سازی این قسمت نیز با توجه به کدهای مربوط به آزمایش هشتم و توضیحات داده­شده در این قسمت به پیوست ارسال می­گردد. خروجی این برنامه مانند زیر خواهد بود.



همانطور که مشاهده می­شود زمان اختصاص داده شده به پردازه­ها بر حسب الویت آن­ها بوده و برای پردازه با الویت 1 هیچ محدودیتی قرار داده نشده­است و لذا این پردازه توانسته در 23 واحد زمانی متوالی (که بیش از base time quantum است) پردازنده را در دست بگیرد.