حل تمرین پنجم درس سیستم‌های عامل

دکتر زرندی

پاییز ۰۰

1- در خصوص انواع فرایندها به سوالات زیر پاسخ دهید.

الف) فرایند فرزند چگونه منابع مورد نیاز خود را تامین می‌کند؟ آیا می‌تواند از منابع والد استفاده کند؟

ب) همانطور که می‌دانید فرایند فرزند ممکن پیش از اتمام اجرا، توسط فرایند والد به پایان برسد. توضیح دهید که فرایند والد به چه دلایلی ممکن است تصمیم بگیرد فرایند فرزند پایان یابد؟

ج) به کد زیر دقت کنید. آیا قطعه مربوط به والد اجرا می‌شود؟ توضیح دهید.

```c
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
int main()
{
pid_t pid;
/* fork a child process */
pid = fork();

if (pid < 0) {/* error occurred */
fprintf(stderr, "Fork Failed");
return 1;
}

else if (pid == 0) {/* child process
printf("Child");
}

else {/* parent process */
wait(NULL);
printf("Parent");
}

```

1- به طور کلی، هنگامی که فرایندی یک فرایند فرزند ایجاد می کند، آن فرایند فرزند به منابع خاصی (مثل زمان اجرا روی CPU، حافظه، فایل ها و دستگاه های I/O) برای اجرا نیاز دارد. یک فرآیند فرزند میتواند منابع مورد نیاز را مستقیماً از سیستم عامل گرفته، یا ممکن است به زیر مجموعه ای از منابع فرایند والد محدود شود.

2- فرایند والد ممکن است اجرای فرایند های فرزند خود را به دلایل مختلفی خاتمه دهد، مانند موارد زیر:

- فرایند فرزند از منابعی که به او اختصاص داده شده، بیش از حد استفاده کرده است. (برای تعیین اینکه آیا این اتفاق افتاده است یا خیر، فرایند والد باید مکانیزمی برای بررسی وضعیت فرزندان خود داشته باشند.)
- وظیفه ای که به فرایند فرزند محول شده دیگر نیاز به انجامش نیست.
- فرایند والد در حال پایان اجرا است و سیستم عامل به فرایند فرزندی که اجرای والد آن پایان یافته، اجازه ی ادامه ی اجرا نمی دهد.

3- بله چراکه با پایان یافتن فرآیند فرزند، فرآیند والد شروع به کار می کند. (علیرغم عدم حضور exec() در فرزند)

2- در یک بافر چرخه‌ای مقدار اشاره‌گر شروع (*in*) برابر 9 و مقدار اشاره‌گر پایان (*out*) برابر با 8 است. فرض کنید این بافر در حالت پر قرار دارد. در این صورت حداقل سایز این بافر چرخه‌ای و همچنین حداکثر تعداد تکه داده (*item*) ای که می‌توان در این حالت در بافر قرار داد را محاسبه کنید. (فرض کنید که پر یا خالی بودن بافر صرفا با پارامترهای *in* و *out* مشخص می‌شود.)

حداقل سایز بافر چرخه ای حالتی است که اشاره گر in در خانه ی انتهایی بافر قرار گرفته باشد (قبل از خانه ی شماره 0) در این صورت حداقل سایز بافر برابر با **10** است و همچنین حداکثر تعداد تکه داده (item) ای که در یک بافر چرخه ای میتوان قرار داد برابر با BufferSize-1 یعنی **9** است: (اشاره گر in در واقع همان start و اشاره گر out همان end است)

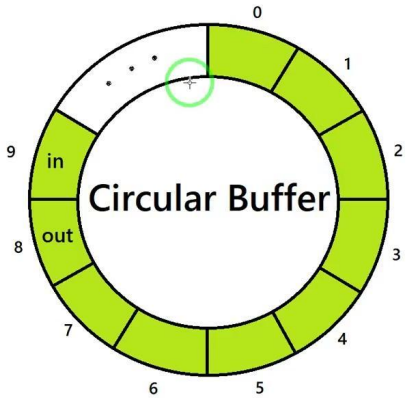
(out+1) % BUFFER_SIZE = In

(8 + 1) % BUFFER_SIZE = 9

BUFFER_SIZE=10 حداقل سایز بافر

Maximum_Item=BUFFER_SIZE - 1

Maximum_Item = 9 حداکثر تعداد تکه داده



(out+1) % BUFFER_SIZE = In
(8 + 1) % BUFFER_SIZE = 9
BUFFER_SIZE=10 حداقل سایز بافر
Maximum_Item=BUFFER_SIZE - 1
Maximum_Item = 9 حداکثر تعداد تکه داده

برای پخش ویدیو VPNنیاز است.

3- همانطور که می‌دانید، یکی از روش‌هایی که فرآیندها برای ارتباط با یکدیگر استفاده می‌کنند حافظه مشترک (*shared memory*) است. در این روش نوع داده‌های حافظه مشترک و محل ذخیره آن‌ها چگونه مشخص می‌شود؟ چگونه تضمین شده است که فرآیندهای مختلف به صورت همزمان در یک مکان یکسان از حافظه مشترک ننویسند؟

فرم داده ها و مکان ذخیره آنها توسط خود این فرآیندها تعیین می شود و تحت کنترل سیستم عامل نیست. فرآیندها همچنین مسئول اطمینان از عدم نوشتن همزمان در یک مکان نیز هستند.

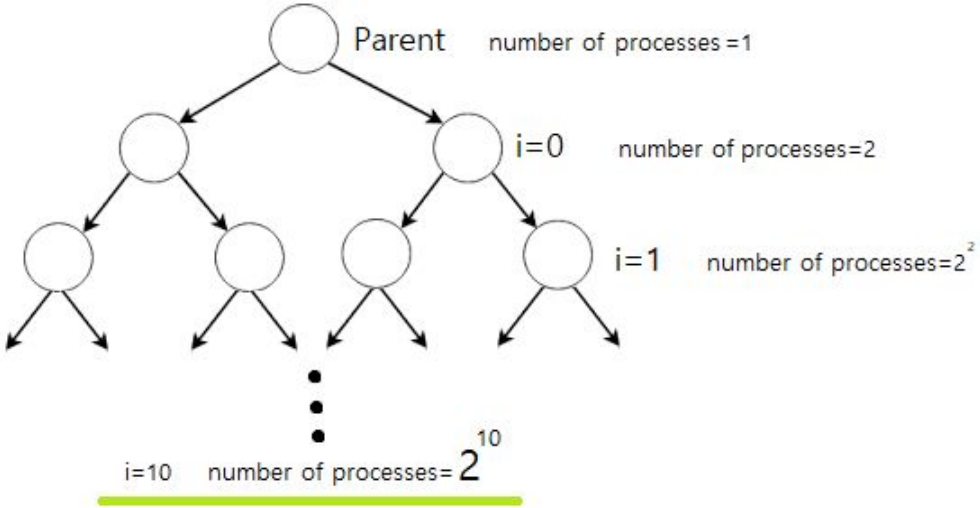
4- سورس کد زیر را در نظر بگیرید:

```
void main(){
    int i;
    for (i=0 ; i<10 ; i++){
        fork();
    }
}
```

الف) در اجرای آن چند فرایند خواهیم داشت؟

ب) چنانچه فرآیند والد بخواهد منتظر پایان یافتن فرزندها باشد این کد چطور باید تغییر کند؟

الف)در ابتدا یک فرایند که فرایند والد است را خواهیم داشت و در هر دور از حلقه یکبار تابع ()fork فراخوانی میشود که در هر فراخوانی، تعداد فرایند ها 2 برابر خواهد شد:



همانطور که در شکل مشاهده می شود 1024 فرایند خواهیم داشت.
(1023 فرایند فرزند و 1 فرایند والد)

برای پخش ویدیو VPNنیاز است.

ب) کد تغییر یافته به صورت زیر است:

(در این صورت فرایند والد پس از هر بار اجرای fork() منتظر پایان یافتن فرایند فرزند خواهد ماند.چک کردن مقدار p که بزرگتر از صفر باشد هم برای **اطمینان از اینکه در فرایند والد باشیم** و سپس wait را انجام دهیم است)

```
void main(){
    int i;
    for (i=0 ; i<10 ; i++){
        p=fork();
        if (p>0){
                wait(NULL);
                }
         }
    }
```