حوری دهش 9821413

سوال 1:

الف)

برای خاتمه دادن به اجرای پروسس فرزند توسط والد می توان از سیستم کال kill استفاده کرد

ب)

علت اول: مدیریت منابع:

ممکن است فرزند منابع زیادی را گرفته باشد و والد بخواهد جلوی اجرای پروسس فرزند را بگیرد مثلا پروسس فرزند حافظه زیادی اشغال کرده است و والد میخواهد از ان پروسس، منابع ازاد شود.

علت دوم: انجام کارهای محدود:

والد ممکن است وظیفه خاصی به پروسس فرزند سپرده باشد و بعد از انجام ان وظیفه، نیازی به ادامه اجرای پروسس فرزند نباشد مثلا از پروسس فرزند خواسته است که یکسری محاسبات انجام بدهد و زمانی که جواب به یک حد خوبی رسید دیگر ادامه ندهد برای همین والد با دستور kill می تواند جلوی این کارو بگیرد

علت سوم: تمام شدن کارهای والد:

ممکن است در یک سیستمی تنها در صورتی که والد زنده است اجازه دهیم فرزند هم وجود داشته باشد پس وقتی که والد کارش تموم شد دیگر نیازی به ان فرزند نیست پس والد می تواند فرزند را kill بکند و خودش هم تمام بشود

سوال 2:

الف)

اگر یک پروسس فرزندی تمام شود و والدش wait را فراخوانی نکرده باشد در این حالت پروسس والد نمی تواند اطلاعات پروسس فرزند را دریافت و پردازش کند پس به پروسس فرزند در این حالت پروسس زامبی می گوییم چون pcbاش وقتی از بین می رود که والد wait را فراخوانی کرده باشد و اگر wait فراخوانی نشده باشد pcb ان پروسس تا مدتی توی سیستم می ماند

ب)

اگر یک والدی wait را فراخوانی نکرده باشد و خودش تمام بشود ولی فرزندش هنوز زنده باشد یعنی اجرای فرزند هنوز تمام نشده باشد در این حالت به پروسس فرزند یتیم گفته میشود چون ان پروسس دیگر والدی ندارد

ج)

برنامه 1 :

در این برنامه پروسس زامبی ایجاد میشود چون هر دو پروسس یعنی پروسس والد و فرزند همزمان با هم اجرا میشوند و پروسس فرزند زودتر از والد تمام میشود در حالی که پروسس والد باید 60 ثانیه منتظر بماند تا تمام شود و چون در پروسس والد اطلاعات فرزند گرفته نمیشود که pcb اش پاک شود پس پروسس فرزند زامبی میشود.

برنامه 2 :

در این برنامه پروسس یتیم ایجاد میشود چون هر دو پروسس یعنی پروسس والد و فرزند همزمان با هم اجرا میشوند و در این حالت پروسس فرزند باید 30 ثانیه منتظر بماند و پروسس والد 20 ثانیه پس پروسس والد زودتر از پروسس فرزند تمام میشود و در این حالت پروسس فرزند یتیم میشود.

سوال 8:

خروجی در Line A برابر با 5 خواهد شد

چون زمانی که پروسس والد، فرزندی ایجاد میکند فضای حافظه فرزند کپی ای از فضای حافظه والد خواهد شد و بعد از اینکه پروسس فرزند ایجاد شد اگر تغییری در هر کدام رخ دهد یعنی تغییری در فضای حافظه خود ایجاد کنند فقط درون انها این تغییر اتفاق می افتد و بر دیگری هیچ تاثیری ندارد یعنی در اینجا که در پروسس فرزند 15 =+ value است فقط این به علاوه 15 در فرزند رخ میدهد و تاثیری بر والد ندارد برای همین خروجی والد همان عدد 5 خواهد شد

سوال 9:

الف)

مقدار a = 5 خواهد شد. چون هنگامی که پروسس فرزند ایجاد میشود فضای حافظه فرزند کپی ای از فضای حافظه والد خواهد بود و بعد از ایجاد فرزند هر تغییری که در فضای حافظه هر کدام رخ دهد بر دیگری تاثیری ندارد یعنی تغییراتی که برای متغییر a در والد رخ می دهد تاثیری بر مقدار a در فرزند نخواد داشت

ب)

تلاش برای خواندن از fd در پروسس فرزند موفقیت امیز نخواهد بود به علت اینکه هنگامی که fork اجرا میشود یک کپی از فایل fd به پروسس فرزند منتقل می شود اما این کپی شامل اطلاعات مربوط به وضعیت فایل یعنی خواندن و نوشتن و.. نمی شود. در پروسس والد بعد از fork فایل fd بسته میشود و این کار باعث میشود که در پروسس فرزند فایل fd بسته باقی بماند و زمانی که در پروسس فرزند دستور خواندن از فایل اجرا میشود این دستور موفقیت امیز نخواهد بود زیرا fd در پروسس فرزند به دیسکریپتور معتبری اشاره ندارد (این اشاره گر به دلیل بسته بودن در پروسس والد، در پروسس فرزند نیز بسته می ماند.)

ج)

با توجه به اینکه پروسس والد قبل از پروسس فرزند زمان بندی میشود و پروسس فرزند برای اولین بار بعد از اجرای کامل پروسس والد زمان بندی میشود پس پروسس والد زودتر از پروسس فرزند تمام میشود در حالی که پروسس فرزند هنوز زنده است ولی دیگر والدی ندارد پس پروسس فرزند یک پروسس یتیم می شود

برای رفع این مشکل در پروسس والد می توانیم از تابع wait یا waitpid استفاده بکنیم:

پس در انتهای والد در کد زیر ما این خط را اضافه میکنیم:

```
int a = 5;
int fd = open(...); // opening a file
int ret = fork();

if (ret > 0) {
        close(fd);
        a = 6;
        // ... (سایر کدها)

        waitpid(ret, NULL, 0);

} else if (ret == 0) {
        printf("a=%d\n", a);
        read(fd, something);
}
```

سوال 10:

طبق توضیحات در سوال های بالایی زمانی که پروسس والد، فرزندی ایجاد میکند فضای حافظه فرزند کپی ای از فضای حافظه والد خواهد شد و بعد از اینکه پروسس فرزند ایجاد شد اگر تغییری در هر کدام رخ دهد یعنی تغییری در فضای حافظه خود ایجاد کنند فقط درون انها این تغییر اتفاق می افتد و بر دیگری هیچ تاثیری ندارد طبق این موضوع نتیجه های زیر را بیان می کنیم:

:Line C

در پروسس فرزند یک ترد ایجاد شده و تابع runner بر روی ان ترد فراخوانی شده است پس مقدار value در اینجا برابر با 5 خواهد شد

:Line P

در پروسس والد مقدار value برابر با صفر است چون تغییراتی که در پروسس فرزند رخ داده است تاثیری بر پروسس والد ندارد و همچنین در این پروسس تابع runner هم فراخوانی نشده است پس مقدار value همان صفر می ماند

سوال 11:

توضیح خطوط مشخص شده در کد:

1. pipe(fd)

این دستور یک پایپ ایجاد میکند که این پایپ دو انتها دارد که به صورت fd[0] و fd[1] در ارایه fd ذخیره میشوند.

2. dup2(fd[1] , 2)

dup2(fd[1], STDOUT FILENO)

دستور dup2(fd[1], 2) جریان خطا یا stderr را به انتهای خروجی پایپ که fd[1] است متصل می کند به عبارت دیگر این دستور، stderr به همان جا که داده ها به پایپ ارسال میشوند متصل میکند و شماره 2 به stderr اختصاص داده شده است و دستور dup2(fd[1], STDOUT FILENO) خروجی استاندارد یا stdout را به انتهای لوله که fd[1] است متصل میکند و این کار باعث میشود هر چیزی که به stdout نوشته میشود به لوله ارسال شود و شماره 1 به stdout اختصاص داده شده است

3. execv(command1, argv1)

این دستور با استفاده از تابع execv دستور اول را اجرا میکند و اگر این دستور اجرا نشود پیام خطا چاپ میشود

4. dup2(fd[0] , 0)

dup2(fd[0], STDIN FILENO)

دستور dup2(fd[0], 0) جریان ورودی یا stdin را به ابتدای پایپ که fd[0] است متصل می کند با این کار هر چی از stdin خوانده میشود از لوله وارد میشود و شماره 0 به stdin اختصاص داده شده است و دستور dup2(fd[0], STDIN FILENO) همان کاری را که dup2(fd[0] , 0) انجام میدهد، انجام میدهد

5. execv(command2, argv2)

این دستور با استفاده از تابع execv دستور اول را اجرا میکند و اگر این دستور اجرا نشود پیام خطا چاپ میشود

6. wait(NULL)

Wait(NULL)

این دستورات منتظر پایان اجرای فرزندان یعنی پروسس های دستور اول و دوم هستند و والد منتظر تمام شدن هر دو فرزند می شود

توضیح کد:

1. pipe(fd): این دستور یک پایپ ایجاد میکند که این پایپ دو انتها دارد که به صورت fd[0] و fd[1] در ارایه fd ذخیره میشوند.

2. pidComm[1]=fork: این دستور با استفاده از تابع fork یک پروسس فرزند ایجاد میکند و والد و فرزند با داشتن یک پایپ به اشتراک اطلاعات می پردازند

3. در بلوک fidComm[1] اگر fidComm[1] برابر با 1- باشد یعنی نمی توان پروسس فرزند را ایجاد کرد و پیام خطا چاپ میشود.

4. در بلوک fidComm[1] اگر fidComm[1] برابر با 0 باشد در پروسس فرزند اول انتهای خروجی (stderr) و خروجی استاندارد (stdout) به پایپ fd[1] متصل می شوند سپس با استفاده از تابع execv دستور اول اجرا میشود و اگر اجرای این دستور موفقیت امیز نباشد پیام خطا چاپ میشود و فرایند خاتمه پیدا میکند.

5. در بلوک fidComm[1] اگر fidComm[1] بزرگتر از 0 باشد در پروسس والد یک فرزند دیگر هم ایجاد میشود حالا اگر مقدار pidComm[0] که مربوط به پروسس جدید است کوچکتر از صفر باشد پروسس فرزند ایجاد نمیشود و پیام خطا چاپ میشود، اگر pidComm[0] برابر با صفر باشد درونش ابتدا انتهای لوله را می بندد سپس stdin را به ابتدای لوله متصل میکند و در نهایت پروسس فرزند دوم با استفاده از تابع exect دستور دوم را اجرا میکند اگر اجرای این دستور موفقیت امیز نباشد پیام خطا چاپ میشود و فرایند خاتمه پیدا میکند.

6. در بلوک fidComm[2] اگر fidComm[2] بزرگتر از 0 باشد در پروسس والد انتهاهای پایپ یا لوله بسته میشوند و در نهایت با استفاده از دستور wait(NULL) منتظر پایان اجرای هر دو پروسس فرزند می شود.