آزمایش ۶ - ارتباط بین پردازه‌ای

۶.۱ مقدمه

در این جلسه از آزمایشگاه مکانیزم‌های مربوط به ارتباط و تبادل پیام بین پردازه‌ها در سیستم عامل لینوکس را خواهیم آموخت.

۶.۱.۱ پیش‌نیازها

انتظار می‌رود که دانشجویان با موارد زیر از پیش آشنا باشند:

۱. نحوه‌ی ایجاد پردازه‌ها در سیستم عامل لینوکس
۲. برنامه‌نویسی به زبان C/C++
۳. دستورات پوسته‌ی لینوکس که در جلسات قبل فرا گرفته شده‌اند.

۶.۲ ارتباط بین پردازه‌ها

در جلسات قبل نحوه‌ی ایجاد پردازه‌های جدید را آموختیم. در این جلسه سعی داریم روش‌های ارتباط میان این پردازه‌ها را بررسی کنیم. مکانیزم‌های متعددی برای تبادل پیام بین پردازه‌ها وجود دارد که در این جلسه دو روش استفاده از Pipe و Signal را بررسی خواهیم کرد. از جمله کاربردهای ارتباط بین پردازه‌ای می‌توان به همگام سازی و انتقال اطلاعات اشاره کرد.

دستورات Pipe برای کاربران پوسته‌ی لینوکس آشنا هستند. برای مثال شما می‌توانید برای مشاهده ی لیست پردازه‌هایی که در آن‌ها کلمه‌ی init وجود دارد از دستور `ps aux | grep init` استفاده کنید. در اینجا دو پردازه به کمک یکPipe به هم متصل شده‌اند. نکته‌ای که در اینجا مهم است آن است که این Pipe ایجاد شده تنها در یک جهت (از پردازه ی اول به پردازه ی دوم) اطلاعات را جابه‌جا می‌کند. به کمک فراخوانی‌های سیستمی می‌توان Pipe های دو سویه و حلقوی نیز ایجاد کرد.

۶.۳ شرح آزمایش

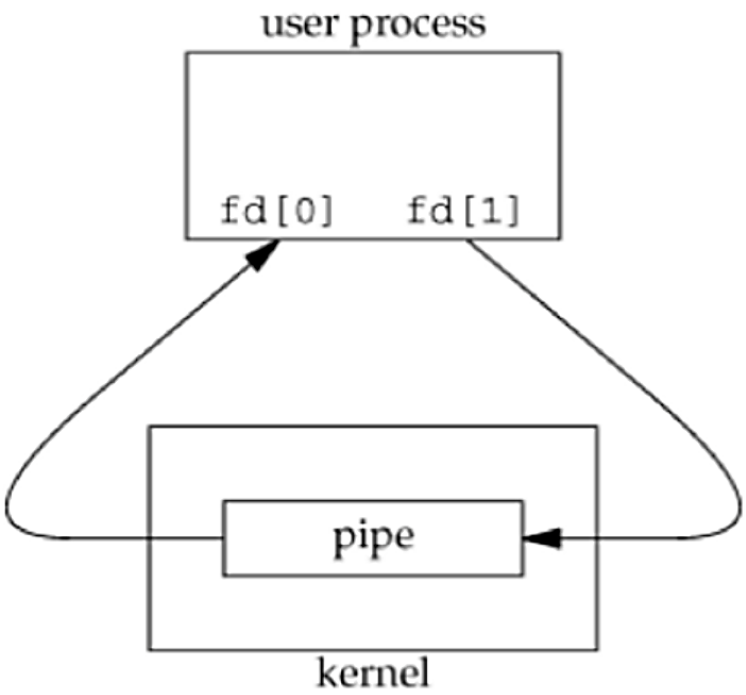
۶.۳.۱ ایجاد یک Pipe یک‌سویه

- (آ) برای ایجاد Pipe یک سویه در سیستم عامل لینوکس از فراخوانی سیستمی pipe استفاده می‌شود. به کمک دستور man 2 pipe خلاصه‌ای از نحوه‌ی کار آن را ملاحظه کنید.
- (ب) به کمک کد زیر یک Pipe ایجاد کنید.

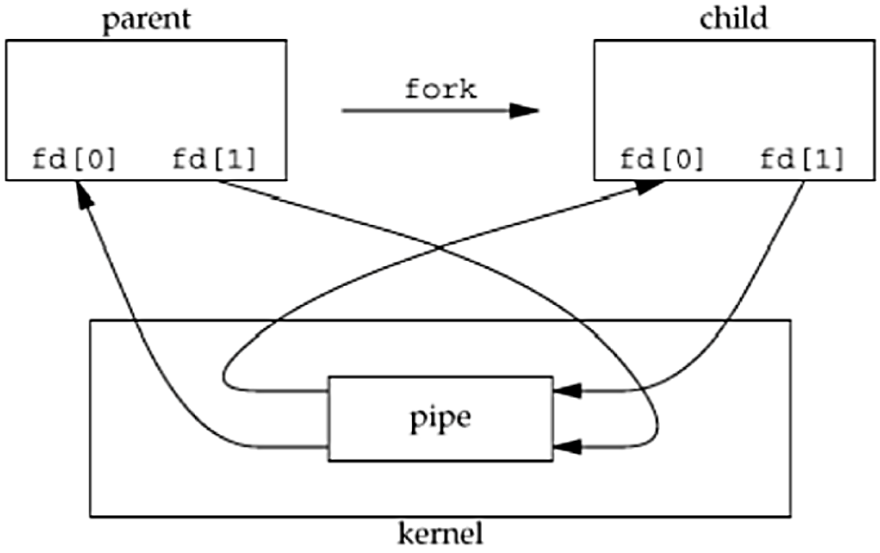
```
int fd[2];
int res = pipe(fd);
```

دستور pipeدر اینجا دو File Descriptor ایجاد می کند (آرایه‌ی fd).یکی از آن‌ها برای خواندن و دیگری برای نوشتن مورد استفاده قرار خواهد گرفت fd[0] برای خواندن و fd[1] برای نوشتن خواهد بود.

- (ج) تا اینجا تنها یک پردازه داریم و می‌توان شمای کلیfd های ایجاد شده را در شکل زیر نشان داد:



هر چیزی که بر روی fd[1]نوشته شود، قابل خواندن با fd[0]خواهد بود. حال توجه کنید که در صورتی که عملیات fork انجام گیرد، پردازه‌ی فرزند، File Descriptorهای پدر را به ارث خواهد برد. بنابراین، بعد از انجام شدن عملیات forkو ایجاد پردازه ی فرزند، ساختار بالابه شکل زیر در خواهد آمد.



مشکل مهمی که در اینجا با آن مواجه هستیم آن است که در صورتی که هر دو پردازه بخواهند بر روی Pipe بنویسند و یا از آن بخوانند، به دلیل اینکه تنها یک بافر مشترک داریم، رفتار سیستم قابل پیش بینی نخواهد بود. در این حالت یک پردازه ممکن است داده‌ای که خودش بر روی Pipe قرار داده است را بخواند! بنابراین نیاز است که یک طرف تنها بر روی Pipe بنویسد و یک طرف تنها از آن بخواند. برای مثال فرض کنید پردازه‌ی فرزند قصد خواندن از Pipe و پردازه‌ی والد قصد نوشتن بر روی آن را دارد. به کمک فراخوانی سیستمی close، پردازه والد fd[0]خود را می‌بندد (زیرا قصد خواندن ندارد) و پردازه‌ی فرزند نیز fd[1]را خواهد بست. به این ترتیب یک ارتباط Half-Duplex بین این دو پردازه ایجاد می‌شود.

فعالیت‌ها

- به کمک توضیحات بالا و استفاده از فراخوانی‌های سیستمی writeو read، جمله‌ی Hello World!را از سمت پردازه‌ی پدر به پردازه ی فرزند منتقل کرده و در پردازه‌ی فرزند آن را چاپ کنید.
- همان‌طور که در جلسات پیش آموختیم، به کمک دستورات خانواده‌ی exec، بعد از انجام forkمی‌توان یک برنامه، مثلا lsرا اجرا نمود. به کمک دستورات dup/dup2برنامه‌ای بنویسید که پردازه‌ی والد دستور lsو پردازه فرزند دستور wcرا اجرا کند و خروجی پردازه ی والد (دستور ls) به عنوان ورودی به پردازه‌ی فرزند داده شود.

- (راهنمایی: یک Pipe ایجاد کنید به نحوی که خروجی پردازه ی والد ورودی Pipe باشد و خروجی Pipe به عنوان ورودی پردازه‌ی فرزند باشد. اینکار با استفاده از دستورات dup/dup2ممکن است).
- (راهنمایی ۲: خروجی برنامه ی lsدر stdoutقرار می‌گیرد که یک File Descriptor با شماره ی ۱ است. ورودی برنامه wcاز stdinاست که یک File Descriptor با شماره ی ۰ است).

? بررسی کنید که چطور ارتباطات تمام دو طرفه بین پردازه‌ها داشته باشیم.

۶.۳.۲ سیگنال‌ها

بعضی اوقات نیاز است که برنامه‌ها بتوانند با برخی از شرایط غیر قابل پیش بینی مواجه شده، آن‌ها را کنترل کنند. برای مثال:

- درخواست بستن برنامه توسط کاربر به وسیله یCtrl + C
- رخ دادن خطا در محاسباتFloating Point
- مرگ پردازه‌ی فرزند

این رخدادها توسط سیستم عامل لینوکس شناخته می‌شوند و سیستم عامل با ارسال یک **سیگنال**، پردازه را از وقوع آن‌ها آگاه می‌سازد. برنامه نویس می‌تواند این سیگنال‌ها را نادیده بگیرد، یا در عوض با نوشتن کد آن‌ها را مدیریت و کنترل نماید.

? به کمک دستور man 7 signal لیستی از سیگنال‌های موجود در سیستم عامل لینوکس را ملاحظه کنید. سیگنال‌های زیر را توضیح دهید.

SIGINT, SIGHUP, SIGSTOP, SIGCONT, SIGKILL

- Information about signals
- Nore information about Two signals: SIGSTOP and SIGCONT

? یک سیگنال ساده، سیگنال Alarm (SIGALRM) است. به کمک دستور manدر مورد آن توضیح کوتاهی ارائه دهید.

? کد زیر به کمک این سیگنال نوشته شده است. آن را اجرا کرده و در مورد کارکرد آن توضیح دهید.

```
#include <stdio.h>
#include <unistd.h>
int main() {
        alarm (5);
        printf ("Looping forever . . . \n");
        while (1);
        printf("This line should never be executed\n");
        return 0;
}
```

? به طور پیش فرض پردازه بعد از دریافت یکی از سیگنال‌های تعریف شده، کشته می‌شود. به کمک تابع sigactionمی‌توان این رفتار را تغییر داد و کد مورد نظر برنامه نویس را اجرا کرد. همچنین یک تابع دیگر به نام pauseوجود دارد که پردازه را تا زمانی که یک سیگنال دریافت کند، متوقف می‌سازد (واضح است که این توابع از فراخوانی‌های سیستمی برای انجام کار خود استفاده می‌کنند). به کمک این دو تابع، برنامه‌ی بالا را به گونه‌ای ویرایش کنید که بعد از دریافت سیگنال SIGALRM از توقف خارج شود و خط آخر را در خروجی چاپ کند.

? برنامه‌ای بنویسید که در صورتی که کاربر کلیدهای Ctrl + C را فشار دهد، برای بار اول خارج نشود و پیامی برای فشار دادن دوباره ی آن‌ها چاپ کند. در دفعه ی دومِ رخ دادن سیگنال، برنامه به پایان برسد.