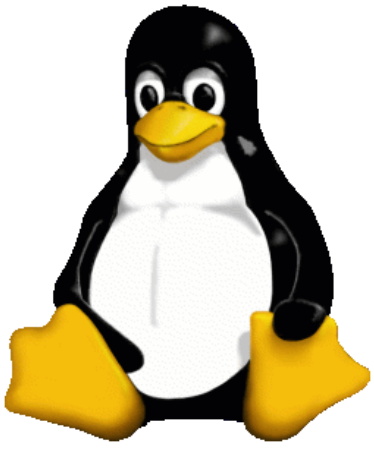
تألیف

دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر

دستورکار آزمایشگاه سیستم‌های عامل

جلسه پنجم: ارتباط بین پردازه‌ای



بهار ۱۳۹۳

مقدمه

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

پیش‌نیازها

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

- نحوه‌ی ایجاد پردازه‌ها در سیستم‌عامل لینوکس (مطالب جلسه‌ی چهارم)
- برنامه‌نویسی به زبان C/C++
- دستورات پوسته‌ی لینوکس که در جلسات قبل فرا گرفته شده‌اند.

ارتباط بین پردازه‌ها

در جلسات قبل نحوه‌ی ایجاد پردازه‌های جدید را آموختیم. در این جلسه سعی داریم روش‌های ارتباط میان این پردازه‌ها 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 جمله‌ی کاربردهای ارتباط بین پردازه‌ای می‌توان به همگام‌سازی و انتقال اطلاعات اشاره کرد.

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

شرح آزمایش

الف) ایجاد یک Pipe یک سویه

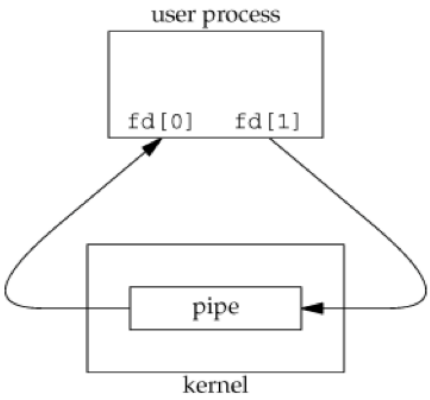
۱. برای ایجاد Pipe یک سویه در سیستم‌عامل لینوکس از فراخوانی سیستمی pipe استفاده می‌شود. به کمک دستور man 2 pipe خلاصه‌ای از نحوه‌ی کار آن ملاحظه کنید.

۲. به کمک کد زیر یک pipe ایجاد کنید:

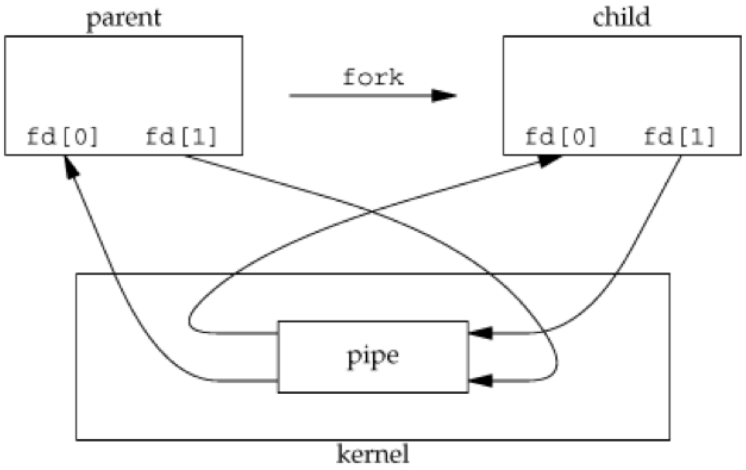
```
int fd[2];
int res = pipe(fd);
```

دستور pipe در اینجا دو File Descriptor ایجاد می‌کند (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.

۳. تا اینجا که تنها یک پردازه داریم، می‌توان شمای کلی fdهای ایجاد شده را به صورت 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 را صورت دهیم، پردازه‌ی فرزند، File Descriptorهای پدر را به ارث خواهد برد. بنابراین بعد از انجام شدن عملیات fork و ایجاد پردازه‌ی فرزند، ساختار بالا به شکل زیر در خواهد آمد:



مشکل مهمی که در اینجا با آن مواجه هستیم آن است که در صورتی هر دو پردازه بخواهند بر روی Pipe بنویسند و از آن بخوانند، به دلیل اینکه تنها یک بافر مشترک داریم، رفتار قابل پیش‌بینی 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 پدر قصد نوشتن بر روی آن را دارد. به کمک فراخوانی سیستمی close، پردازه‌ی پدر 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.

به کمک توضیحات بالا و استفاده از فراخوانی‌های سیستمی write و read، جمله‌ی Hello World را از سمت پردازه‌ی پدر به پردازه‌ی فرزند منتقل کرده و در پردازه‌ی فرزند آن را چاپ کنید.

✓ فعالیت‌ها

- همان‌طور که در جلسه‌ی پیش آموختیم، به کمک دستورات خانواده‌ی exec، بعد از انجام fork می‌توان یک پردازه، مثلاً ls، را اجرا نمود. به کمک دستورات dup/dup2 برنامه‌ای بنویسید که پردازه‌ی پدر دستور ls و پردازه‌ی فرزند دستور wc را اجرا کند و خروجی پردازه‌ی پدر (که دستور ls است) به عنوان ورودی به پردازه‌ی فرزند داده شود. *راهنمایی: یک pipe ایجاد کنید و به نحوی خروجی‌ای که در حالت عادی به Standard Output نوشته می‌شود را به آن منتقل کنید. پردازه‌ی فرزند نیز باید به جای خواندن از ورودی استاندارد از pipe ورودی خود را بخواند. این کار به کمک دستورات dup/dup2 امکان پذیر است.*
- چگونه ارتباطات تمام دوطرفه بین پردازه‌ها داشته باشیم؟

ب) سیگنال‌ها

بعضی اوقات نیاز است که برنامه‌ها بتوانند با برخی از شرایط غیر قابل پیش‌بینی مواجه شده و آن‌ها را کنترل کنند؛ برای مثال:

- درخواست بستن برنامه توسط کاربر به وسیله‌ی Ctrl+C
- رخ‌دادن یک خطا در محاسبات Floting Point
- مرگ پردازه‌ی فرزند

این رخ‌دادها توسط سیستم‌عامل لینوکس شناخته می‌شوند و سیستم‌عامل با ارسال یک «سیگنال»، پردازه را از وقوع آن‌ها آگاه می‌سازد. برنامه‌نویس می‌تواند این سیگنال‌ها را نادیده بگیرد، یا در عوض با نوشتن کد مخصوص آن‌ها را مدیریت و کنترل نماید. به این ترتیب، برنامه‌نویس می‌تواند برنامه‌ی خود را به نحوی تغییر دهد که مثلاً با دریافت کلیدهای Ctrl+C بسته نشود.

۱. به کمک دستور man 7 signal لیستی از سیگنال‌های موجود در سیستم‌عامل لینوکس را ملاحظه کنید. برخی از آن‌ها را به دلخواه انتخاب و در گزارش خود با توضیح مختصری بیاورید.
۲. یک سیگنال ساده، سیگنال Alarm است. به کمک man در مورد آن توضیح کوتاهی ارائه دهید.
۳. کد زیر به کمک این سیگنال نوشته شده است؛ آن را اجرا کرده و در مورد کارکرد آن توضیح دهید:

```
#include <stdio.h>

int main()
{
	alarm(5);
	printf("Looping forever...\n");
	while(1);
	printf("This line should never be executed.\n");
	return 0;
}
```

۴. به طور پیش‌فرض پردازه بعد از دریافت یکی از سیگنال‌های تعریف شده، کشته می‌شود. به کمک فراخوانی سیستمی signal می‌توان این رفتار را تغییر داد و کد موردنظر برنامه‌نویس را اجرا کرد. همچنین یک فراخوانی سیستمی دیگر به نام pause وجود دارد که پردازه را تا زمانی که یک سیگنال دریافت کند، متوقف می‌سازد. به کمک این دو تابع، برنامه‌ی بالا را به‌گونه‌ای ویرایش کنید که بعد از دریافت سیگنال alarm که با SIGALRM شناخته می‌شود، از توقف خارج شود و خط آخر را در خروجی چاپ کند.

✓ فعالیت‌ها

- برنامه‌ای بنویسید که در صورتی که کاربر کلیدهای Ctrl+C را فشار دهد، برای بار اول خارج نشود ولی در دفعه‌ی دوم برنامه به پایان برسد.