دانشگاه علم و صنعت

دانشکده مهندسی کامپیوتر

درجه تحصیلی: کارشناسی

گزارشکار تمرین5 OS

گردآورنده:

پرنیان شاکریان - 99400064

استاد:

دکتر انتظاری

سال تحصیلی: خرداد ۱۴۰۲

خلاصه:

در تمرین پنجم سیستم عامل قصد داریم مبحث Inter-process communication (IPC) را بررسی کنیم. اگر بخواهیم به طور خلاصه توضیح دهیم، (IPC) به تکنیک‌هایی اشاره دارد که توسط process برای تبادل اطلاعات، هماهنگ کردن فعالیت‌ها و همگام‌سازی عملیات آنها استفاده می‌شود. (IPC) اجازه می دهد تا processهای در حال اجرا بر روی یک سیستم با یکدیگر ارتباط برقرار کرده و مشارکت کنند.

روش های (IPC):

۱. Pipes: شکل ساده‌ای از IPC را ارائه می دهند که در آن داده از طریق یک کانال ارتباطی یک طرفه بین فرآیندها منتقل می شود. یک فرآیند، داده ها را در Pipe نوشته و فرآیند دیگر آن را می خواند.
۲. Shared Memory: به چندین فرآیند اجازه میدهد تا به بخش حافظه‌ای که در آنها مشترک است دسترسی داشته باشند. فرآیندها می توانند با mapping کردن ناحیه‌ای از حافظه مشترک در فضای آدرس خود، آن را خوانده و بنویسند و اشتراک گذاری داده ها را فراهم کنند. (مکانیسم‌های سمافور یا mutex اغلب برای هماهنگ کردن دسترسی به حافظه مشترک استفاده می‌شوند.)
۳. signals: راهی برای processها به منظور ارسال notification یا قطع سیگنال به processهای دیگر است. از سیگنال‌ها می‌توان برای مدیریت انتقال اطلاعات بین processها استفاده کرد، اما از نظر داده‌هایی که می‌توانند حمل کنند محدود هستند.

شرح کلی: برای حل سوالات تمرین در ابتدا لازم است کلیه کتابخانه‌های مورد نیاز را وارد کنیم. کلیه کتابخانه های مورد استفاده در این تمرین نمایش داده شده است.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
```

2. Write a program in which a parent forks two children. The first child writes “My process id is X, my parent’s id is Y, and my sibling’s id is Z” (where X, Y, and Z are the process ids). The program should produce no other output. For this purpose, you are asked to use the pipe system call.

در ابتدای کد باید کتابخانه‌های مورد نیاز خود را وارد کنیم که به ترتیب توابع و typeهای مورد نیاز برای عملیات ورودی/خروجی، تخصیص حافظه و API سیستم عامل را ارائه میدهند. (کتابخانه‌های مورد نیاز در ابتدای داک اشاره شده است)

تابع main به عنوان نقطه ورودی برنامه اعلام می شود. در داخل تابع یک آرایه صحیح fd با اندازه ۲ برای نشان دادن توصیفگرهای pipe و دو متغیر نوع pid_t برای ذخیره processes id فرزندان اعلام شده‌اند.

در ادامه تابع pipe برای ایجاد یک pipe فراخوانی شده و مقدار بازگشتی آن برای خطاها بررسی می شود. این تابع یک pipe یک طرفه ایجاد کرده که از دو توصیف کننده فایل تشکیل شده است که در صورت موفقیت ۰ و در صورت شکست ۱- را برمی گرداند.

- fd[۰] برای خواندن از pipe
- fd[۱] برای نوشتن در pipe

اولین process فرزند با فراخوانی fork ایجاد می شود. تابع fork ابتدا یک process فرزند جدید ایجاد می کند، سپس process id فرزند را به process والد و ۰ را به process فرزند برمی گرداند. اگر فراخوانی با شکست مواجه شود، ۱- را برمی گرداند. داخل اولین process فرزند، یک process فرزند دیگر ایجاد شده و دوباره مقدار بازگشتی آن برای خطا بررسی می شود. اگر فراخوانی fork موفق به ایجاد دومین process فرزند شود، کد بلوک if (pid2 == 0) اجرا می شود. (این بلوک دومین process فرزند را نشان میدهد.)

process فرزند دوم:

۱. character buffer با اندازه ۱۰۰ برای ذخیره داده‌های خوانده شده از pipe اعلام میشود.
۲. داده‌ها از pipe fd[0] در بافر، توسط تابع read خوانده و در خروجی چاپ میشوند.
۳. انتهای pipe خوانده شده fd[0]، با استفاده از close(fd[0]) بسته می شود.
٤. اگر فراخوانی fork در اولین فرزند، موفق به ایجاد دومین فرزند شود، بلوک else اجرا میشود.

process فرزند اول:

۱. character buffer با اندازه ۱۰۰ برای ذخیره داده‌های نوشته شده در pipe اعلام میشود.
۲. idهای اولین process فرزند، والد و دومین process فرزند در بافر چاپ میشوند.
۳. داده‌ها در بافر با استفاده از تابع write روی pipe fd[1] نوشته می شوند.
٤. انتهای pipe نوشته شده fd[1]، با استفاده از close(fd[1]) بسته می شود.

پس از آنکه اولین process فرزند نوشتنش را داخل pipe تمام کرد، process والد wait(NULL) را فراخوانی می‌کند تا process فرزند terminate شود. در آخر تابع main برای نشان دادن اجرای موفق برنامه EXIT_SUCCESS را برمیگرداند. خروجی‌ها: