دانشکده مهندسی کامپیوتر

درس سیستم‌های عامل

تمرین سری دوم

مدرسان ..... دکتر رضا انتظاری ملکی، دکتر وحید ازهری

تیم طراح ..... محسن رحیمی – سینا علی‌نژاد

تاریخ انتشار ۱۴۰۲/۰۸/۱۰

تاریخ تحویل ۱۴۰۲/۰۸/۲۰

در رابطه با تمرین

- این تمرین شامل مبحث:
  - Thread
  - Process

می‌باشد.

- نمره این تمرین از ۱۰۰ می‌باشد و بارم هر سوال روبه‌روی آن نوشته شده است.
- به هیچ وجه تمرینی را از دیگران کپی نکنید. در صورت مشاهده تقلب و کپی در تمرینات، نمره هر دو طرف صفر در نظر گرفته می شود.

۱- جستجوی فایل (۲۵ نمره)

در این سوال شما باید یک عمل سرچ را به صورت multi-thread پیاده سازی کنید.

ورودی برنامه شما به صورت زیر است:

```
search <start_dir> <search_query>
```

خروجی برنامه شما نیز به این صورت است:

```
[+] Found:
./path/to/file

[+] Found:
./path2/to/file

...
```

برنامه باید به شکل multi-thread عمل جستجو را انجام دهد (به این گونه که هر دایرکتوری در یک thread جدید بررسی شود.) وگرنه نمره ای به تمرین تعلق نمی‌گیرد.

برای خواندن محتویات یک directory می‌توانید از کتابخانه dirent.h استفاده کنید.

کد زیر فایل ها و directory های موجود در directory فعلی را چاپ می‌کند:

```
#include<stdio.h>
#include<dirent.h>

int main() {
    DIR *dir = opendir("./");
    if (dir == NULL) {
     //raise error
    }

    struct dirent* entry;
    while ((entry = readdir(dir)) != NULL) {
     if (entry->d_type == DT_DIR) {
         printf("dir: %s\n", entry->d_name);
     } else {
         printf("file: %s\n", entry->d_name);
     }
    }

    closedir(dir);
}
```

برای حل این سوال همین مقدار اطلاعات از dirent.h کافی است.

۲- شمارنده کلمات (۲۵ نمره)

در این سوال شما باید به صورت multi-process چندین عبارت را در یک فایل جستجو کنید.

ورودی برنامه باید به صورت زیر باشد:

```
search <file_to_search> <query1> <query2> <query3> ….
```

خروجی برنامه به صورت زیر است:

```
query1: <occurrences of query1>
query2: <occurrences of query2>
query3: <occurrences of query3>
…
Total occurrences: <total occurrences of all queries>
```

برای پیدا کردن تعداد رخداد هر query باید یک پروسس جداگانه ایجاد شود. و در انتها جمع همه رخداد ها محاسبه شود و در خروجی نمایش داده شود.

۳- عملیات Convolution با استفاده از Multi-Processing (۲۵ نمره)

در این تمرین، می‌خواهیم عملیات Convolution را با استفاده از Multi-Processing انجام دهیم. کد اولیه در اختیار شما قرار داده شده است و فقط قسمت‌هایی که مربوط به عملیات اصلی می‌شود، کامنت گذاری شده است تا شما آن را کامل کنید. در انتها می‌توانید مدت زمان انجام عملیات را مشاهده کنید. همچنین یک فایل دیگر قرار داده شده است که عین همین عملیات را بدون استفاده از مالتی پراسس انجام می‌دهد و مدت زمان را گزارش می‌دهد. برای ابعاد کوچک و متوسط ماتریس، روش بدون Multi-Process سریعتر انجام می‌شود، می‌توانید بگویید چرا؟ مقادیر خیلی بزرگ‌تر را امتحان کنید تا ببینید چه زمانی روش Multi-Process بهتر عمل می‌کند و نتیجه را گزارش دهید. برای ساخت ماتریس با ابعاد بزرگ می‌توانید از این لینک استفاده کنید.

نکته: بخش‌هایی که از عبارت NULL و TODO استفاده شده است، باید توسط شما کامل شود.

نکته: در این تمرین، به جای سیستم کال fork از vfork استفاده کنید. این لینک تفاوت این دو تابع را بررسی می‌کند. به طور کلی بخواهیم بگوییم، در vfork، process های مختلف که توسط process اصلی فراخوانی می‌شوند، فضای آدرس دهی یکسانی دارند، در نتیجه هر تغییری در متغیرهای یک process بر روی متغیرهای process دیگر هم اعمال میشود.

کامنت‌ها را با دقت بخوانید تا ابهامی برای سوال باقی نماند.

۴- ساخت Shell با استفاده از زبان C (۲۵ نمره)

در این تمرین، قصد داریم یک shell ساده را با استفاده از زبان C پیاده‌سازی کنیم. اکثر فایل‌های کد برای این تمرین برای شما قرار داده شده است. برای ساخت یک shell چندین مرحله وجود دارد:

- خواندن ورودی از طرف user (read_line): کد این قسمت در فایل read_line.c قرار داده شده و نیازی به پیاده‌سازی توسط شما نیست.
- جدا کردن اجزای ورودی یا همان parse (split_line): این فایل نیز قرار داده شده است.
- اجرای دستور وارد شده (execute_args): بخشی از کدهای این فایل قرار داده شده و مابقی را شما باید پیاده‌سازی کنید.
- همه موارد بالا در فایل main.c فراخوانی می‌شوند.

راهنمایی: برای اجرا کردن برنامه از دستور زیر استفاده کنید:

```
gcc main.c read_line.c split_line.c execute_args.c own_exit.c own_cd.c
```

راهنمایی: از سیستم کال execvp برای اجرای دستور در ترمینال استفاده کنید.

- این تابع را باید درون یک process جدا صدا بزنید وگرنه برنامه پس از اولین دستور خاتمه پیدا می‌کند. در مورد دلیل این اتفاق جستجو کنید و نتیجه را در گزارش بنویسید.
- همچنین برخی از دستورات ترمینال مانند exit و cd توسط این تابع قابل انجام نیست. در مورد دلیل این نیز تحقیق کنید و نتیجه را در گزارش بنویسید. دستور exit قبلا در فایل own_exit پیاده‌سازی شده است. شما دستور cd را درون فایل own_cd پیاده‌سازی کنید. (راهنمایی: از تابع chdir می‌توانید استفاده کنید) برای این کار باید این دستور و تابع مربوط به آن را به لیست مربوطه در فایل execute_args اضافه کنید.

شما فقط در دو فایل execute_args.c و own_cd.c کد خواهید زد، بنابراین کامنت‌های درون این دو فایل را با دقت بخوانید.