به نام خداوند بخشنده مهربان

سیستم‌های عامل

دانشکده مهندسی کامپیوتر

محمدعلی میرزایی و محدثه میربیگی
پاییز ۱۴۰۲

تمرین دوم

اسلایدهای ۳ و ۴ و ۵ تاریخ تحویل: ۱۵ آبان ۱۴۰۲

پرسش ۱ (۲۵ نمره) درستی یا نادرستی جملات زیر را مشخص کنید. برای جملات نادرست دلیل بیاورید:

(آ) context switch همواره به طور ناخواسته و در زمان نامشخص رخ می‌دهد.

(ب) دو ریسه (thread) A و B که مربوط به یک پردازه (process) هستند، همواره می‌تواند از استک (stack) یکدیگر داده بخوانند یا در آن بنویسند.

(ج) با فراخوانی سیستم‌کال execv در لینوکس می‌توان یک پردازه جدید ساخت.

(د) یک پردازه می‌تواند دو file descriptor متفاوت داشته باشد، (در جدول file descriptor همان پردازه) که به یک file description باز شده‌ی یکسان در کرنل اشاره می‌کنند.

(ه) فرض کنید یک برنامه‌ی multithread نوشته‌اید که ۱۰۰۰ ریسه (thread) تولید می‌کند که همگی می‌خواهند داده‌هایی از یک آرایه‌ی Share شده بخوانند. برای آن‌که مطمئن باشید هیچ race conditionای بین این تردها به‌وجود نمی‌آید، باید یک روش سینکرونایزیشن مناسب پیاده‌سازی کرد.

(و) یک پردازه (process) اجازه ندارد تغییری در فضای آدرس‌دهی یک پردازه‌ی دیگر به‌وجود بیاورد.

(ز) زمان انتظار Round Robin به‌طور میانگین کم‌تر از FCFS است.

(ح) Round Robin و FCFS به preemption نیاز ندارند.

(ط) کارآیی حافظه نهان در Round Robin نسبت‌به FCFS بهبود خواهد یافت.

(ی) اگر یک ریسه (thread)، یک file descriptor را ببند، آن file descriptor برای همه ریسه‌های داخل سیستم مستقل از این که متعلق به چه پردازه‌ای هستند بسته خواهد شد.

(ک) در یک پردازه چند ریسه‌ای (multi thread process)، دسترسی به متغیری که در پشته مخصوص یک ریسه ذخیره شده است، thread-safe است.

(ل) الگوریتم SRTF بهینه‌ترین الگوریتم زمان‌بندی است که می‌توان در سیستم‌عامل پیاده‌سازی کرد.

پرسش ۲ (۱۰ نمره) در یک context switch بین دو ریسه در هر حالت زیر چه چیزهایی باید ذخیره و بازیابی شوند؟

(آ) دو ریسه متعلق‌به یک پردازه‌اند.

(ب) دو ریسه متعلق‌به دو پردازه‌ی مختلف‌اند.

پرسش ۳ (۱۰ نمره امتیازی) فرض کنید در سروری زمان سرویس دهی هر درخواست ۲۰ میلی ثانیه باشد. به توجه به این فرض به سوالات زیر پاسخ دهید:

(آ) توضیح دهید در صورتی که میانگین نرخ رسیدن درخواست‌ها برابر با ۶۰ درخواست در ثانیه باشد، تاخیر صف به چه مقداری میل می‌کند؟

(ب) توضیح دهید اگر فاصله بین رسیدن درخواست ها مقدار ثابت ۳۰ میلی ثانیه باشد، و در زمان صفر ۱۰ درخواست در صف باشند، هنگامی که سیستم به حالت پایدار می‌رسد، تاخیر صف چقدر خواهد بود؟

(ج) در حالت توصیف شده در قسمت ب، میزان بهره‌وری سرور در زمان طولانی تقریبا چه مقدار خواهد بود؟

پرسش ۴ (۲۰ نمره) در یک سیستم، ۳ پردازه cpu-bound به نام‌های A و B و C و یک پردازه io-bound به نام D در زمان‌های مختلفی که در جدول نشان داده شده، آزاد می‌شوند. همچنین زمان تعویض متن در این سیستم قابل چشم‌پوشی است. سیستم مبتنی بر قواعد زیر عمل می‌کند:

- کوانتوم زمانی در این سیستم برای اجرای هر وظیفه، ۶ میلی‌ثانیه است.
- عدد بزرگ‌تر در ستون اولویت وظایف نشان‌دهنده اولویت بالاتر است.
- وظیفه io-bound هر بار ۱ میلی‌ثانیه اجرا می‌شود و سپس یک درخواست io به یک دستگاه خروجی ارسال می‌کند و این چرخه تا زمان اتمام وظیفه تکرار می‌شود.
- دستگاه خروجی به هر درخواست io در زمان ۲ میلی‌ثانیه پاسخ می‌دهد.
- کوانتوم زمانی مربوط به وظیفه io-bound با هر درخواست io ریست نمی‌شود. بنابراین برای مثال اگر هنگامی که هنوز ۴ ثانیه از کوانتوم زمانی وظیفه باقی مانده و یک درخواست io داده می‌شود، این کوانتوم زمانی ذخیره شده و در دور بعدی که وظیفه اجرا می‌شود، کوانتوم زمانی آن به جای ۶ از همان ۴ محاسبه می‌شود.
- وظایفی که از صف io خارج می‌شوند، وارد صف ready می‌شوند. وظایفی که هنوز کوانتوم زمانی پایان‌نیافته از قبل دارند به ابتدای صف و وظایفی که باید کوانتوم زمانی جدیدی را آغاز کنند به انتهای صف منتقل می‌شوند.

جدول ۱: مشخصات وظایف سیستم

وظیفه	زمان ورود	مدت زمان اجرا	اولویت
A	۱۲	۲۸	۴
B	۱۹	۳۴	۷
C	۲	۳۶	۴
D	۷	۹	۷

- در صورتی که در لحظه‌ای خاص از دوره زمان‌بندی، نیاز به اجرای همزمان دو یا چند عمل باشد، ترتیب اعمال باید به صورت زیر باشد:
  - وظیفه تازه به دست سیستم رسیده شده را در صف ready قرار بدهید.
  - صف ready را برای انتخاب وظیفه بعدی که باید اجرا شود با الگوریتم مورد نظر بررسی کنید.
  - وظیفه‌ای که از CPU خارج می‌شود را در صف io یا ready مربوطه قرار دهید.
  - وظیفه‌ای که از صف io خارج می‌شود را در صف ready قرار دهید.

برای هر یک از ۳ حالت الگوریتم زمان‌بندی خواسته‌شده، تمام مراحل زمان‌بندی و تغییرات وظیفه در حال اجرا را در قالب یک جدول به صورت زیر بنویسید. همچنین turnaround time را برای هر یک از وظایف و برای میانگین همه وظایف محاسبه کنید.

- زمان‌بندی round-robin بدون در نظر گرفتن اولویت‌ها
- زمان‌بندی round-robin با در نظر گرفتن preemption و اولویت وظایف در لحظه ورود
- زمان‌بندی FCFS با در نظر گرفتن اولویت و بدون preemption

پرسش ۵ (۱۰ نمره) قطعه کد زیر را در نظر بگیرید

```c
int main() {
    printf("Starting main\n");
    int file_fd = open("test.txt", O_WRONLY | O_TRUNC | O_CREAT, 0666);
    dup2(file_fd, STDOUT_FILENO);
    pid_t child_pid = fork();
    if (child_pid != 0) {
        wait(NULL);
        printf("In parent\n");
    } else {
        printf("In child\n");
    }
    printf("Ending main: %d\n", child_pid);
}
```

خروجی استاندارد و محتوای فایل test.txt چه خواهد بود؟ توضیح دهید. فرض کنید هیچ سیسکالی فیل نمیشد و pid پردازه فرزند ۶۶۶۶ است.

پرسش ۶ (۱۰ نمره) کد زیر را در نظر بگیرید:

```c
int global;

void *foo(void *arg) {
    printf("%p\n", &global);
    printf("%p\n", &foo);
    printf("%p\n", &arg);
    printf("%p\n", arg);
    return NULL;
}

int main() {
    void *hmem = malloc(1);
    for (int i = 0; i < 3; i++) {
        pthread_t thread;
        pthread_create(&thread, NULL, foo, hmem);
    }
}
```

از بین خطوط `printf` در تابع `foo` کدام خطها خروجی یکسانی در تمام ریسه‌ها دارند؟

پرسش ۷ (۱۰ نمره) در قطعه کد زیر، امکان چاپ حداکثر چند رشته متمایز از اعداد در خروجی وجود دارد؟ توضیح دهید.

```c
void *myThread(int arg)
{
    printf("%d", arg);
    return 0;
}

int main()
{
    int i;
    pthread_t tids[5];
    printf("1");

    for (i = 0; i <= 4; i++)
        pthread_create(&tids[i], NULL, myThread, i+2);

    for (i = 0; i < 4; i++)
        pthread_join(tids[i], NULL);

    printf("7");
    return 0;
}
```

پرسش ۸ (۱۵ نمره) در هر یک از تکه‌کدهای زیر تمام موارد درست را انتخاب کنید. در تمام بخش‌ها فرض کنید:

- فراخوانی‌های **open**, **fopen**, **fork**, **pthread_create**, **malloc**, **realloc** همواره با موفقیت انجام می‌شوند.
- فراخوانی‌های **read**, **write**, **dup**, **dup2** در صورتی که یک file descriptor معتبر برای آن‌ها ارائه شود، با موفقیت انجام می‌شوند.
- تمامی سرآیندهای موردنیاز از لایبرری استاندارد C در کد **#include** شده‌اند.
- پیش از اجرای تمامی برنامه‌ها، محتوای فایل **file.txt** خالی است.
- تمامی تردها در نهایت انجام پیش می‌روند. هیچ فرض اضافه‌تری در رابطه با زمانبند نکنید!

(آ) کدام یک از موارد زیر می‌توانند محتوای **file.txt** بعد از terminate شدن تمام پردازه‌ها باشند؟

```c
int main(int argc, char** argv) {
    if (fork() == 0) {
        int fd1 = open("file.txt", O_WRONLY);
        write(fd1, "a", 1);
    } else {
        int fd2 = open("file.txt", O_WRONLY);
        write(fd2, "b", 1);
    }
}
```

[] (empty) [] a [] b [] aa [] ab [] ba [] bb

(ب) کدام یک از موارد زیر می‌توانند محتوای **file.txt** بعد از terminate شدن تمام پردازه‌ها باشند؟

```c
char buffer;
int main(int argc, char** argv) {
    int fd = open("file.txt", O_WRONLY);
    if (fork() == 0) {
        buffer = 'a';
    } else {
        buffer = 'b';
    }
    write(fd, &buffer, 1);
}
```

[] (empty) [] a [] b [] aa [] ab [] ba [] bb

(ج) کدام یک از موارد زیر می‌توانند محتوای **file.txt** بعد از terminate شدن برنامه‌ی زیر باشند؟

```c
int fd;
void* helper(void* arg) {
    write(fd, "a", 1);
}
int main(int argc, char** argv) {
    fd = open("file.txt", O_WRONLY);
```

```
    pthread_t thread;
    pthread_create(&thread, NULL, helper, NULL);
    write(fd, "b", 1);
}
```

[] (empty) [] a [] b [] aa [] ab [] ba [] bb

(د) کدام یک از موارد زیر می‌توانند محتوای **file.txt** بعد از terminate شدن برنامه‌ی زیر باشند؟

```
int fd;
void* helper(void* arg) {
    write(fd, "a", 1);
    pthread_exit(NULL);
}
int main(int argc, char** argv) {
    fd = open("file.txt", O_WRONLY);
    pthread_t thread;
    pthread_create(&thread, NULL, helper, NULL);
    write(fd, "b", 1);
    pthread_exit(NULL);
}
```

[] (empty) [] a [] b [] aa [] ab [] ba [] bb

(ه) کدام یک از موارد زیر می‌توانند محتوای **file.txt** بعد از terminate شدن برنامه‌ی زیر باشند؟

```
int fd;
char buffer;
void* helper(void* arg) {
    buffer = 'a';
    write(fd, &buffer, 1);
    pthread_exit(NULL);
}
int main(int argc, char** argv) {
    fd = open("file.txt", O_WRONLY);
    pthread_t thread;
    pthread_create(&thread, NULL, helper, NULL);
    buffer = 'b';
    write(fd, &buffer, 1);
    pthread_exit(NULL);
}
```

[] (empty) [] a [] b [] aa [] ab [] ba [] bb