操作系统作业 3

1. What are the two models of interprocess communication? What are the strengths and weaknesses of the two approaches?

   答：**共享内存**和**消息传递**。

   共享内存：

   优点：仅在建立共享内存区时使用系统调用，之后的访问无需借助内核，更快速

   缺点：多进程同时访问共享区域时可能引发冲突，需解决；在物理内存不相邻的分布式系统上不易实现；多处理器中，共享数据在多个缓存之间迁移还会造成“一致性问题”等

   消息传递：

   优点：不会引起冲突；分布式系统上易于实现

   缺点：频繁使用系统调用，内核的介入消耗时间

2. What are the benefits of multi-threading? Which of the following components of program state are shared across

threads in a multithreaded process?

a. Register values　　b. Heap memory

c. Global variables　　d. Stack memory

答：优点如下：

响应性好：部分线程阻塞或执行冗长操作时，程序中的其它线程仍可以执行，从而增加对用户的访问程度

资源共享：线程默认共享它们所属进程的内存和资源

开销小：线程更为轻量级，创建、切换远比进程迅速。使用多线程对系统资源使用更少

可伸缩性：多线程使得一个进程可以同时运行在多个 CPU 上

上述 4 部分中， b. Heap memory 和 c. Global memory会被该进程的所有线程共享

3. Consider the following code segment:

```
pid t pid;

pid = fork();

if (pid == 0) { /* child process */
```

```
        fork();
        thread create( . . .);
    }
    fork();
```

a. How many unique processes are created?

答：第 1、2个fork()均执行了1次，第 3个fork()执行了 3次，新创建了 1+1+3 = 5 个进程

b. How many unique threads are created?

答：每个新进程至少包含 1 个线程；有 2 个进程又分别额外创建了 1 个线程，故新创建了 5+2= 7 个线程

4. The program shown in the following figure uses Pthreads. What would be the output from the program at LINE C and LINE P?

```c
#include <pthread.h>
#include <stdio.h>

int value = 0;
void *runner(void *param); /* the thread */

int main(int argc, char *argv[])
{
pid_t pid;
pthread_t tid;
pthread_attr_t attr;

   pid = fork();

   if (pid == 0) { /* child process */
      pthread_attr_init(&attr);
      pthread_create(&tid,&attr,runner,NULL);
      pthread_join(tid,NULL);
      printf("CHILD: value = %d",value); /* LINE C */
   }
   else if (pid > 0) { /* parent process */
      wait(NULL);
      printf("PARENT: value = %d",value); /* LINE P */
   }
}

void *runner(void *param) {
   value = 5;
   pthread_exit(0);
}
```

Figure: C program for Question 4.

答：子进程创建线程执行 runner()，父进程进行 wait 系统调用，且 value 为 int 型全局变量，仅在 fork()时拷贝。故答案为：

LINE C: CHILD: value = 5

LINE P: PARENT: value = 0

5. What are the differences between ordinary pipe and named pipe?

   普通管道：单向通信；由父进程创建，与其创建的子进程之间进行通信；进程完成通信终止后，管道不复存在
   命名管道：通信可以是双向的，且父子关系不是必须的；多个进程都可以使用同一命名管道进行通信；通信进程完成后，命名管道继续存在，直到它被显式地从文件系统中删除

6. What is race condition? Which property can guarantee that race condition will not happen?

   答：

   race condition：多个进程并发访问和操作同一共享数据，并且执行结果与特定访问顺序有关

   进程同步，保证进程访问共享资源时是**互斥**的

7. The first known correct software solution to the critical-section problem for two processes was developed by Dekker. The two processes, P0 and P1, share the following variables:

boolean flag[2]; /* initially false */

int turn;

The structure of process Pi (i == 0 or 1) is shown in the following Figure; the other process is Pj (j == 1 or 0). Prove that the algorithm satisfies all three requirements for the critical-section problem.

```
do {
   flag[i] = true;

   while (flag[j]) {
      if (turn == j) {
         flag[i] = false;
         while (turn == j)
            ; /* do nothing */
         flag[i] = true;
      }
   }

      /* critical section */

   turn = j;
   flag[i] = false;

      /* remainder section */
} while (true);
```

Figure: The structure of process Pi for Question 7.

答:

(1)**互斥成立**：由上述代码可知，只有当 flag[j] == false 时，Pi 进程才能进入临界区。则若 Pi 与 Pj 同时在临界区内执行，则有 flag[i] == flag[j] == false；而在 Pi 进入临界区前，其 flag[i]值已被设置为 true，Pj 同理，显然矛盾。故 Pi 与 Pj 不可能同时在临界区内执行，满足互斥条件

(2)**progress**：如果进程 i 不准备进入临界区，

则 flag[i]==false

显然不会影响进程 j 进入临界区。该条件成立

(3)**有限等待**：当进程 i 结束临界区任务后，会将 turn 交给 j 进程，同时将 flag[i]设为 false，此后 j 进程便可以跳出 while 循环或者直接进入临界区。J 进程结束时同理。故满足有限等待条件

8. Can strict alternation and Peterson's solution sastify all the requirements as a solution of the critical-section problem? Please explain why.

答：

Strict alternation: 违反了要求#3："在关键部分之外运行的任何进程都不应阻塞其他进程。"

例如，当进程 0 从临界区退出后，将 turn 重设为 1；此时及时进程 1 没有进入临界区，进程 0 也会被阻止进入临界区

Peterson's solution：满足了所有要求

(1)互斥：对于 0 号进程，只有当 flag[1] == false 或

turn == 0 时才能进入临界区。若两个进程同时进入临界区，

则 flag[0] == flag[1] == true；而 turn 的值要么为 0，要

么为 1，则必有一个进程仍陷入 while 循环中无法进入临

界区，出现矛盾。故两个进程不可能同时进入临界区，互斥条件满足

(2)显然该方案没有依赖于进程在临界区中执行的时间

(3)当进程 0 不准备进入临界区时，若 flag[0] == false,此时进

程 1 可以进入临界区；如果 0 进程已设置 flag[0] == true 且

仍陷在 while 循环，则此时 turn == 1，进程 1 仍可进入临界区

(4)有限等待：当进程 0 退出临界区时，将 flag[0]值为 false,

允许进程 1 进入临界区；此时若进程 1 正陷在 while 循环中

等待，则只需再判断循环条件 1 次即可跳出，之后便进入临界区。满足有限等待条件

9. What is semaphore? How to use semaphore to

implement section entry and section exit (no busy

waiting)? Please give the code.

答：信号量是一个数据类型，除了初始化外只能通过两个标准原子操作down()、up()（PV操作）来访问，即修改过程应不可分割地执行。

使用信号量来完成 entry 段和 exit 段代码：

```
typedef struct{
    int value;
    struct process *list;
}semaphore;

void down(semaphore *s){
    disable_interrupt();
    while(s->value == 0){
        enable_interrupt();
        special_sleep();
        disable_interrupt();
    }
    s->value = s->value - 1;
    enable_interrupt();
}

void up(semaphore *s){
    disable_interrupt();
    if(s->value == 0)
        special_wakeup();
    s->value = s->value + 1;
    enable_interrupt();
}
```

10.What is deadlock? List the four requirements of deadlock.

答:
死锁：两个及以上进程无限等待一事件，而该事件只能由等待进程之一来产生；多个进程相互等待，互不相让，导致所有进程无限期等待。

4 个条件:

(1)互斥：某一资源在同一时间只能被一个进程占有

(2)占有和等待：死锁中的进程既占有其它进程所需要的资源，又在等待请求其它进程所占有的资源

(3)无优先权：某一资源只有在占有它的进程完成任务后才会被释放，而不能被其它进程抢占

(4)循环等待：死锁中进程的占有、等待构成一个环形关系