OS hw2

1. T1

> Excluding the initial parent process, how many child processes are created by the program shown in Figure 1?

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

int main()
{
    int i;

    for (i = 0; i < 4; i++)
       fork();

    return 0;
}
```

Figure 1: Program for Question 1.

一共有$2^0 + 2^1 + 2^2 + 2^3 = 15$个子进程

2. T2

> Explain the circumstances under which the line of code marked printf (“LINE J”) in Figure 2 will be reached. Please also explain the functionality of the wait() system call

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
pid_t pid;

    /* fork a child process */
    pid = fork();

    if (pid < 0) { /* error occurred */
       fprintf(stderr, "Fork Failed");
       return 1;
    }
    else if (pid == 0) { /* child process */
       execlp("/bin/ls","ls",NULL);
       printf("LINE J");
    }
    else { /* parent process */
       /* parent will wait for the child to complete */
       wait(NULL);
       printf("Child Complete");
    }

    return 0;
}
```

当fork成功且execlp失败时，printf（“LINE J”）才会被执行

wait（）：对 wait() 的调用会阻止调用进程，直到它的一个子进程退出或收到信号为止

- 通知父进程子进程结束运行
- 告诉父进程子进程是如何结束的，wait返回结束的子进程的PID给父进程

3. T3

Using the program in Figure 3, identify the values of pid at lines A, B, C, and D. (Assume that the actual pids of the parent and child are 600 and 603, respectively.

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
pid_t pid, pid1;

    /* fork a child process */
    pid = fork();

    if (pid < 0) { /* error occurred */
       fprintf(stderr, "Fork Failed");
       return 1;
    }
    else if (pid == 0) { /* child process */
       pid1 = getpid();
       printf("child: pid = %d",pid); /* A */
       printf("child: pid1 = %d",pid1); /* B */
    }
    else { /* parent process */
       pid1 = getpid();
       printf("parent: pid = %d",pid); /* C */
       printf("parent: pid1 = %d",pid1); /* D */
       wait(NULL);
    }

    return 0;
}
```

Figure 3: Program for Question 3.

当fork函数调用成功时，父进程的返回值是子进程的pid，子进程的返回值为0。

在 A 和 B 中，子进程。A 输出函数 fork 的返回值：0;

B 输出子进程的 pid：603。

在 C 和 D 中，父进程。C 输出函数fork的返回：603;

D 输出父进程的 pid：600

所以：

A： 0

B： 603

C： 603

D： 600.

4. T4

Using the program shown in Figure 4, explain what the output will be at lines X and Y

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

#define SIZE 5

int nums[SIZE] = {0,1,2,3,4};

int main()
{
int i;
pid_t pid;

   pid = fork();

   if (pid == 0) {
      for (i = 0; i < SIZE; i++) {
         nums[i] *= -i;
         printf("CHILD: %d ",nums[i]); /* LINE X */
      }
   }
   else if (pid > 0) {
      wait(NULL);
      for (i = 0; i < SIZE; i++)
         printf("PARENT: %d ",nums[i]); /* LINE Y */
   }

   return 0;
}
```

Figure 4: Program for Question 4.

LINE X： 0, -1, -4, -9, -16

LINE Y：0, 1, 2, 3, 4

5. T5

> For the program in Figure 5, will LINE Y be executed, and explain why.
>
> ```
> int main(void) {
>     printf("before execl ...\n");
>     execl("/bin/ls", "/bin/ls", NULL);
>     printf("after execl ...\n");      /*LINE: Y*/
>     return 0;
> }
> ```
>
> Figure 5: Program for Question 5.

分两种情况：

1. 如果exec（）成功执行，则不会返回原程序并退出，因此不会执行LINE Y
2. 如果exec（）没有成功执行，则会返回到原程序，LINE Y被执行

6. T6

> Explain what data will be stored in user-space and kernel-space memory for a prosess.

- 在用户空间中：全局变量、局部变量、代码和常量
- 在内核空间中：PCB （包括进程状态、程序计数器、CPU 寄存器、CPU 脱落信息、内存管理信息、I/O 状态信息和记帐信息）、系统调用代码等

7. T7

> Explain why “terminated state” is necessary for processes

- 终止状态的存在是子进程将信息反馈到父进程的一种机制。子进程退出后，父进程应读取子进程的退出状态，即检查子进程的运行结果
- 进程的终止状态可以显示进程的结束方式。它不仅可以通过发送带有终止状态信息的信号来唤醒父进程，还可以通知父进程和子进程终止信息
- 如果父函数中存在wait（）函数，则父进程将暂停其进程并等待子进程变为终止状态并唤醒它。因此，可以控制父进程和子进程的执行顺序

8. T8

> Explain what a zombie process is and how to eliminate a zombie process (i.e., remove its PCB entry from kernel)

- 僵尸进程是指在计算机操作系统中已经完成执行但仍在进程表中具有PCB，且其父进程尚未调用wait()系统调用来检索其退出状态的一种进程类型；当进程完成执行时，通常会向其父进程发送一个终止信号，表示其资源可以释放。然后，父进程可以使用wait()系统调用检索终止的子进程的退出状态。然而，如果父进程未调用wait()或在调用wait()之前终止，则子进程将成为僵尸进程

- 要消除僵尸进程，需要父进程检索子进程的退出状态，读取到退出状态代码，才能从内核空间中回收PCB数据结构，从而退出僵尸状态

9. T9

> Explain the key differences between exec() system call and normal function call

关键的区别在于 exec（） 系统调用可以用其参数指定的程序替换进程的代码空间，可以导致执行新程序而不会返回

普通函数调用不会改变执行程序，并返回原程序

10. T10

> 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

- benifits：
    1. 提高程序的性能和响应速度：多线程可以将程序分解为多个并行执行的子任务，利用多个 CPU 核心同时处理任务，从而提高程序的运行效率和响应速度。
    2. 改善程序的交互性：多线程可以让程序在后台执行任务，同时保持与用户界面的响应，从而改善程序的交互性和用户体验。
    3. 便于编写复杂的程序：多线程可以使程序更加模块化和可维护，减少代码的复杂性和耦合度，从而更容易编写复杂的程序。
- 选择：b、c

11. T11

> 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?

b. How many unique threads are created?

- a. 6个进程
- b. 2个线程

12. T12

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);
}
```

LINE C：5

子进程调用runner，将value设置为5

LINE P：0

在完成子进程之后，父进程中存在的全局变量的值仍然为0

13. T13

> What are the two abstract models of IPC? Explain their pros and cons.

IPC有两个抽象模型：共享内存和消息传递。

- 共享内存模型是指进程之间共享同一块内存空间，从而实现通信和数据共享

  优点：效率高，因为直接在内存中读写数据，没有复制和传输数据的开销

  缺点：需要解决同步和互斥的问题

- 消息传递模型是指进程之间通过传递消息来进行通信，每个进程都有自己的地址空间，消息传递是通过操作系统内核来完成的

  优点：可靠性高，因为操作系统负责确保消息的正确性和可靠性，不需要考虑同步和互斥问题

  缺点：效率较低，因为需要在内核空间和用户空间之间进行上下文切换，并且需要复制和传输数据

14. T14

> What are the differences between ordinary pipe and named pipe?

- 普通管道适用于同一台计算机上的进程间通信，而命名管道可以被多个进程共享，并且可以在不同计算机上的进程间进行通信
- 普通管道是一次性的，只能被创建它的进程和它的子进程使用，而命名管道是持久的，并且可以被多个进程共享。