# 操作系统作业二

1. Including the initial parent process, how many 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;
}
```

答：共创建 2^4　= 16 个子进程

2. Explain the circumstances under which the line of code marked printf ("LINE J") in Figure 2 will be reached.

```c
#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;
}
```

Figure 2: Program for Question 2.

答：若程序执行至 printf（“LINE　J”），则说明：

（1）父进程成功创建子进程

（2）子进程执行系统调用 execlp 时，没有找到路径为“/bin/ls”的程序

3. 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 2600 and 2603, respectively.)

```c
#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.

答：
A: 变量 pid 为 fork()对子进程的返回值，为 0
B: 变量 pid1 为子进程 PID，值为 2603
C: 变量 pid 为子进程 PID，值为 2603
D: 变量 pid1 为 getpid()获取的父进程 PID,值为 2600

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

```c
#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.

答：父进程调用 wait(NULL)， 等待子进程先执行；

子程序的 pid 变量为 0，故 X 行输出结果为：

CHILD: 0 CHILD: -1 CHILD: -4 CHILD: -9 CHILD: -16

且父、子进程中指向 Global 区的指针 nums 值共享，故父进程输出结果（即 Y 行输出结果）为：

PARENT: 0 PARENT: -1 PARENT: -4 PARENT: -9 PARENT: -16

5. For the program in Figure 5, will LINE X be executed, and explain why.

```
int main(void) {
    printf("before execl ...\n");
    execl("/bin/ls", "/bin/ls", NULL);
    printf("after execl ...\n");      /*LINE: X*/
    return 0;
}
```

Figure 5: Program for Question 5.

答：

不会被执行。进行 execl 系统调用后，原来包含 LINE: X 编译结果的代码段会被/bin/ls 的代码段替换，在 ls 程序的 return/exit 语句执行后该进程直接结束。

6. Explain why “terminated state” is necessary for processes.

答：

进程执行 exit()系统调用后，进入 terminated 状态。

该状态下： 此进程再也不会被执行；进程用户空间下内存已被全部清空；内核中的大部分资源也已被操作系统回收，只将部分部分信息保留，如进程结束状态等。

进而父进程通过访问子进程的 entry 可以获得子进程的相关信息，据此可以获得子进程是否正常结束，或者出现了何种异常等情况

7. Explain what a zombie process is and when a zombie process will be eliminated (i.e., its PCB entry is removed from kernel).

答：

子进程执行完成或被显式终止之后，并在其父进程回收其资源前，该进程进入 terminated 状态，成为 “僵尸进程”

子进程先于父进程退出时，会发出 SIGCHILD 信号以唤醒父进程。父进程接收到该信号之后，会在 PCB 里自己的 entry

中查寻相应的“signal handlers”例程。父进程默认不对SIGCHILD 信号进行响应，但若父进程进行了 wait()、waitpid()等系统调用，则父进程接受到该信号后，会进行相应的一些处理，之后将 PCB 里子进程的 entry 中所有信息均清除掉。