

操作系统作业 2

要求：独立完成，严禁抄袭，按时提交。

提交截止日期：4.10（周日）18:00。

提交方式：提交 PDF 版本，PDF 命名规则为学号_姓名_作业 2， 比如 PB200110000_张三_作业 2.pdf。

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;
}
```

Figure 1: Program for Question 1.

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

```
#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.

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.)

```
#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.

4. 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.

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.

6. Explain why “terminated state” is necessary for processes.
7. Explain what a zombie process is and when a zombie process will be eliminated (i.e., its PCB entry is removed from kernel).
8. Explain what data will be stored in user-space and kernel-space memory for a process.
9. Explain the key differences between `exec()` system call and normal function call.

10. 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

11. 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?

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

```
#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);
}
```

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