Rüzgar Ayan 21801984

2020-2021 SPRING CS342 HOMEWORK 2 Report

Question 1.

The code below runs a for loop k times and for each step of the iteration, the number of processes will double since each child process will call the fork() too. Because of this reason, after the for loop ends, we will have a total of 2^k processes and their pid's will be printed. This code assumes a correct argument input from the user, so it doesn't check them. Also, wait statements could be added if we want the parent to wait before returning.

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

int main(int argc, char *argv[])
{
    int i;
    char* a = argv[1];
    int k = atoi(a);
    for (i = 0; i < k; ++i) {
        fork();
    }

    int pid = getpid();
    printf("%d\n", pid);

    return 0;
}
```

The example execution below creates 2^3=8 processes total.

```
zzlawlzz@zzlawlzz-VirtualBox:~/Desktop$ ./q1 3
34407
34411
34408
34409
34410
34412
34413
34414
```

Question 2.

To find the definition of struct task_struct, I have used the grep command on the kernel that I've downloaded before.

```
zzlawlzz@zzlawlzz-VirtualBox:~/Downloads/linux-5.10.12$ grep -r 'struct
task_struct {' *
include/linux/sched.h:struct task_struct {
tools/include/linux/lockdep.h:struct task_struct {
tools/testing/selftests/bpf/progs/test_core_retro.c:struct task_struct {
tools/testing/selftests/bpf/progs/test_core_reloc_kernel.c:struct task_struct
{
tools/testing/selftests/bpf/progs/strobemeta.h:struct task_struct {};
```

In the first result above, /include/linux/sched.h, I've found the definition and some of the attributes were:

1. struct thread_info thread_info;
2. volatile long state;
3. unsigned int flags;
4. unsigned int ptrace;
5. int on_cpu;
6. struct __call_single_node wake_entry;
7. unsigned int cpu;
8. unsigned int wakee_flips;
9. int recent_used_cpu;
10. int wake_cpu;

Question 3.

The table below shows the number of processes created in each iteration of the for loop. Other than the main process, 10 processes are created according to the pseudocode.

When	Newly Created Processes	Exited Processes	Remaining Total Processes
Beginning	1(main process)	0	1
for loop i=0	1	1	1
for loop i=1	1	0	2
for loop i=2	2	2	2
for loop i=3	2	0	4
for loop i=4	4	4	4

The results can also be confirmed by executing the following C program.

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

```c
#include <unistd.h>

int main(int argc, char *argv[])
{
    int i;
    for (i = 0; i < 5; ++i) {
        printf("Created in %d\n", i);
        int n = fork();
        if (n == 0) {
            if (i % 2 == 0) {
                printf("Exited in %d\n", i);
                exit(0);
            }
        }
    }

    return 0;
}
```

Question 4.

This pseudocode prints the integers **100 once** and **250 three times** because for the process where x gets 200, it gets replaced by the **ls** execution and never reaches the print statement. This can also be confirmed by executing the C program below.

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

int main(int argc, char *argv[]) {
    int x = 100;
    if (fork() == 0) {
        x += 100;
        if (fork() == 0) {
            x += 50;
            printf("%d\n",x);
        }
        else {
            execl("/bin/ls", "ls", NULL); // execute the ls program
            printf("%d\n",x);
        }
        printf("%d\n",x);
    }
    printf("%d\n",x);

    return 0;
}
```

Question 5.

```c
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>

int main(int argc, char *argv[]) {

    if (fork() == 0) //First child process is created here
    {
        execl("/bin/ps", "ps", "aux", NULL);
    }
    else
    {
        if (fork() == 0) //Second child process is created here
        {
            execl("/bin/ls", "ls", "-al", NULL);
        }
    }

    //Waiting for the two child processes to end
    wait(NULL);
    wait(NULL);

    printf("Parent process ends\n");

    return 0;
}
```

Question 6.

For this question, I mostly used the sample code given for message queues since the task was almost the same. I also used the header file “shareddefs.h” but I changed the length of astr char array from 64 to 128 since the message we were trying to send was exceeding 64 bytes.

```c
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <mqueue.h>
#include <string.h>

#include "shareddefs.h" //From the example on the moodle

void senderChild(mqd_t mq)
{
    struct item item;
    int n;
```

```
    item.id = 1;
    strcpy(item.astr, "I hear and I forget. I see and I remember. I do and I
understand.");

    n = mq_send(mq, (char *) &item, sizeof(struct item), 0);

    if (n == -1) {
        perror("mq_send failed\n");
        exit(1);
    }

    printf("mq_send success\n");
}

void receiverChild(mqd_t mq)
{
    struct mq_attr mq_attr;
    struct item* itemptr;
    char *bufptr;
    int buflen;
    int n;

    mq_getattr(mq, &mq_attr);
    buflen = mq_attr.mq_msgsize;
    bufptr = (char *) malloc(buflen);

    n = mq_receive(mq, (char *) bufptr, buflen, NULL);
    if (n == -1) {
        perror("mq_receive failed\n");
        exit(1);
    }

    itemptr = (struct item *) bufptr;
    printf("mq_receive success, the following message is received:\n%s\n",
itemptr->astr);
}

int main(int argc, char *argv[]) {

    mqd_t mq;
    mq = mq_open(MQNAME, O_RDWR | O_CREAT, 0666, NULL);

    if (mq == -1) {
        perror("can not create msg queue\n");
        exit(1);
    }
    printf("mq created, mq id = %d\n", (int) mq);

    if (fork() == 0) //First child process is created here
    {
        senderChild(mq);
        exit(0);
    }
    else
    {
        if (fork() == 0) //Second child process is created here
```

```
        {
            receiverChild(mq);
            exit(0);
        }
    }

    //Waiting for the two child processes to end
    wait(NULL);
    wait(NULL);

    return 0;
}
```

```
zzlawlzz@zzlawlzz-VirtualBox:~/Desktop/hw2$ ./q6
mq created, mq id = 3
mq_send success
mq_receive success, the following message is received:
I hear and I forget. I see and I remember. I do and I understand.
```

Question 7.

Below program takes two arguments as the input and output file names and performs the desired operation.

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

int main(int argc, char *argv[]) {

    if (argc < 3)
    {
        printf("Needs at least 2 arguments\n");
        exit(1);
    }

    char readbuf[1];
    char writebuf[2];
    int n;
    int readfile = open(argv[1], O_RDONLY);
    int writefile = open(argv[2], O_WRONLY | O_CREAT, 0644);

    n = read(readfile, readbuf, 1);
    while (n != 0) {
        writebuf[0] = readbuf[0];
        writebuf[1] = readbuf[0];

        write(writefile, writebuf, 2);
```

```
        n = read(readfile, readbuf, 1);
    }

    close(readfile);
    close(writefile);

    return 0;
}
```