CS342

Operating Systems

Homework #2

Mustafa Yaşar

Question 1.

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

int main() {
        int k, ab; /* k must be greater than or equal to 5 */
        printf("Enter an integer greater than or equal to 5: ");
        scanf("%d", &k);
        pid_t parent, leftChild, rightChild;
        parent = getpid();      /* Parent is the root of the tree */
        printf("Id of the root: %d\n", parent);
        for (int i = 1; i < k; i++) {
                leftChild = fork();
                if ( leftChild != 0 ) {
                        rightChild = fork();
                }
                if ( leftChild != 0 && rightChild != 0 ) {
                        break;
                }
                printf("My id: %d       My parent id: %d\n", getpid(), getppid());
        }
        waitpid(leftChild, &ab, 0);
        waitpid(rightChild, &ab, 0);
        return 0;
}
```

Question 2.

Cpu_init

state

stack

usage

flags

ptrace

sched_class

tasks

exit_state

exit_code

Question 3.

The program creates 10 processes.

The root process creates 5 processes. The process (i1) that is the child of the root process when i = 1 creates 3 processes. The process that is the child of the root process when I = 3 creates 1 process. The child of the process (i1) creates 1 process.

5 + 3 + 1 + 1 = 10 processes in total.

Question 4.

The program executes the ls command and prints 250 for 3 times.

Question 5.

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

#include <sys/wait.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>


int main () {
    pid_t leftChild;

    pid_t rightChild;

    int lr;

    leftChild = fork();


    if ( leftChild != 0 ) {

        rightChild = fork();

    }


    if ( leftChild == 0 && rightChild != 0) {

        /* Left child executes here */

        printf("My id: %d, my parent id: %d\n", getpid(), getppid());

        char *argv[3] = {"ps", "aux", NULL};

        execv("/bin/ps", argv);

        printf("My id: %d, my parent id: %d\n", getpid(), getppid());

    }


    if ( leftChild != 0 && rightChild == 0 ) {

        /* Right child executes here */

        printf("My id: %d, my parent id: %d\n", getpid(), getppid());

        char *argv[3] = {"ls", "-al", NULL};

        execv("/bin/ls", argv);

    }

    waitpid(leftChild, &lr, 0);

    waitpid(rightChild, &lr, 0);

    return 0;

}
```

Question 6.

```
#include <sys/msg.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <string.h>

struct message_queue {
        long messageType;     /* Must be > 0 */
        char text[100];                 /* Message data */
} msg_queue;

int main () {
        pid_t leftChild;
        pid_t rightChild;
        int lr;
        leftChild = fork();
        key_t key = ftok("MessageQueue", 16);
        msg_queue.messageType = 1;
        int messageId = msgget(key, 0666 | IPC_CREAT); /* Creates a message queue */

        if ( leftChild != 0 ) {
                rightChild = fork();
        }

        if ( leftChild == 0 && rightChild != 0) {
                /* Left child executes here */
```

```
		/* Left child writes and sends the message */
		printf("My id: %d, my parent id: %d\n", getpid(), getppid());


		strcpy(msg_queue.text, "I hear and I forget. I see and I remember. I do and I 
understand");
		msgsnd(messageId, &msg_queue, sizeof(msg_queue), 0);
	}


	if ( leftChild != 0 && rightChild == 0 ) {
		/* Right child executes here */
		/* Right child receives and reads the message*/
		printf("My id: %d, my parent id: %d\n", getpid(), getppid())
		msgrcv( messageId, &msg_queue, sizeof(msg_queue), 1, 0);
		printf("Received message is: %s\n", msg_queue.text);
	}
	waitpid(leftChild, &lr, 0);
	waitpid(rightChild, &lr, 0);
	return 0;
}
```

Question 7.

```
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#include <unistd.h>
int main() {


	int inputDescriptor = open("input.txt", O_RDONLY); /* Open the file with read only 
flag */
```

```
	int outputDescriptor = open("output.txt", O_WRONLY | O_CREAT, 0644); /* OPen the
file with write only flag */


	int flag;


	char buffer[1];


	while ( (flag = read (inputDescriptor, buffer, 1)) > 0 ) {
		write(outputDescriptor, buffer, flag);
		write(outputDescriptor, buffer, flag);
	}
	close(inputDescriptor);
}
```