Daffodil International University

Department of Software Engineering

Faculty of Science and Information Technology

Semester: Fall-2017

Course Code: SWE232 **Course Title: Operating System Lab**

Section: A, B **Course Teacher: Dr. Md Mostafijur Rahman**

LAB EXERCISE 2

Submission Due Date: 09-11-2017

Objectives:

1. Ability to Create Processes with fork()
2. Ability to use Process-Related API Functions
3. Ability to Raising and Catching Signals
4. Ability to create Message Queues
5. Ability to Send and Receive Messages

This lab introduces the GNU/Linux process model. It defines elements of a process, how processes communicate with each other, and how to control and monitor them. This lab complete with exercises, sample of applications to illustrate each technique. However, it is necessary to remember and consult the lab report text in order to understand the potential differences between the developed and the existing implementations.

Figure 1 shows a piece of code to get process id information.

```
1: #include <stdio.h>
2: #include <unistd.h>
3: #include <sys/types.h>
4
5: int main()
6: {
7: pid_t myPid;
8: pid_t myParentPid;
9: gid_t myGid;
10: uid_t myUid;
11:
12: myPid = getpid();
13: myParentPid = getppid();
14: myGid = getgid();
15: myUid = getuid();
16:
17: printf( “my process id is %d\n”, myPid );
18:
19: printf( “my parent’s process id is %d\n”, myParentPid );
20:
21: printf( “my group id is %d\n”, myGid );
22:
23: printf( “my user id is %d\n”, myUid );
24:
25: return 0;
26: }
```

Figure 1: ProcessID.c

Q1: Correct the code in Figure 1.
Q2. Explain the role of the routines getpid(), getppid(), getgid() and getuid()
Q3: Explain the output of the code briefly.

Figure 2 shows to create parent and child process.

```
1: #include <sys/types.h>
2: #include <unistd.h>
3: #include <errno.h>
4:
5: int main()
6: {
7: pid_t ret;
8: int status, i;
9: int role = -1;
10:
11: ret = fork();
12:
13: if (ret > 0) {
14:
15: printf("Parent: This is the parent process (pid %d)\n",
16: getpid());
17:
18: for (i = 0 ; i < 10 ; i++) {
19: printf("Parent: At count %d\n", i);
20: sleep(1);
21: }
22:
23: ret = wait( &status );
24:
25: role = 0;
26:
27: } else if (ret == 0) {
28:
29: printf("Child: This is the child process (pid %d)\n",
30: getpid());
31:
32: for (i = 0 ; i < 10 ; i++) {
33: printf("Child: At count %d\n", i);
34: sleep(1);
35: }
36:
37: role = 1;
38:
39: } else {
40:
41: printf("Parent: Error trying to fork() (%d)\n", errno);
42:
43: }
44:
45: printf("%s: Exiting...\n",
46: ((role == 0) ? "Parent" : "Child"));
47:
48: return 0;
49: }
```

Figure 2: SampleProcess.c

Q4: Correct the code in the Figure 2.
Q5: Explain the output of the code briefly.
Q6: Maximum how many child process able to create using your PC.

Figure 3 shows how to catch signal

```
1: #include <stdio.h>
2: #include <sys/types.h>
3: #include <signal.h>
4: #include <unistd.h>
5:
6: void catch_ctlc( int sig_num )
7: {
8: printf( "Caught Control-C\n" );
9: fflush( stdout );
10:
11: return;
12: }
13:
14: int main()
15: {
16:
17: signal( SIGINT, catch_ctlc );
18:
19: printf("Go ahead, make my day.\n");
20:
21: pause();
22:
23: return 0;
24: }
```

Figure 3: Sigcatch.c

Q7: Correct the code in the Figure 3.
Q8: Explain the output of the code briefly.

Figure 4 shows an example to create own shell prompt to execute shell commands.

Q7. Correct the code.
Q8. Explain the output.

```
1: #include <sys/types.h>
2: #include <sys/wait.h>
3: #include <unistd.h>
4: #include <stdio.h>
5: #include <stdlib.h>
6: #include <string.h>
7:
8: #define MAX_LINE 80
9:
10: int main()
11: {
12: int status;
13: pid_t childpid;
14: char cmd[MAX_LINE+1];
15: char *sret;
16:
17: while (1) {
18:
19: printf("mysh>");
20:
21: sret = fgets( cmd, sizeof(cmd), stdin );
23: if (sret == NULL) exit(-1);
24:
25: cmd[ strlen(cmd)-1] = 0;
26:
27: if (!strncmp(cmd, "bye", 3)) exit(0);
28:
29: childpid = fork();
30:
31: if (childpid == 0) {
32:
33: execlp( cmd, cmd, NULL );
34:
35: } else if (childpid > 0) {
36:
37: waitpid( childpid, &status, 0 );
38:
39: }
40:
41: printf("\n");
42:
43: }
44:
45: return 0;
46: }
```

Figure 4: MyShell.c

Figure 5 shows an example to create a message queue for IPC.

Q9. Correct the code.
Q10. Explain the output.

```
1: #include <stdio.h>
2: #include <sys/msg.h>
3: #define MAX_LINE 80
4:
5: #define MY_MQ_ID 111

6: typedef struct {
7: long type; // Msg Type (> 0)
8: float fval; // User Message
9: unsigned int uival; // User Message
10: char strval[MAX_LINE+1]; // User Message
11: } MY_TYPE_T;

12: int main()
13: {
14: int msgid;
15:
16: /* Create the message queue with the id MY_MQ_ID */
17: msgid = msgget( MY_MQ_ID, 0666 | IPC_CREAT );
18:
19: if (msgid >= 0) {
20: printf( "Created a Message Queue %d\n", msgid );
21: }
22:
23: return 0;
24: }
```

Figure 5: Create a message queue (msgcreate.c)

Figure 6 shows an example to send message to another process.

Q11. Correct the code.
Q12. Explain the output.

```
1: #include <stdio.h>
2: #include <sys/msg.h>
3: #define MAX_LINE 80
4:
5: #define MY_MQ_ID 111

6: typedef struct {
7: long type; // Msg Type (> 0)
8: float fval; // User Message
9: unsigned int uival; // User Message
10: char strval[MAX_LINE+1]; // User Message
11: } MY_TYPE_T;

5: int main()
6: {
7: MY_TYPE_T myObject;
8: int qid, ret;
9:
10: /* Get the queue ID for the existing queue */
11: qid = msgget( MY_MQ_ID, 0 );
12:
13: if (qid >= 0) {
14:
```

```
15: /* Create our message with a message queue type of 1 */
16: myObject.type = 1L;
17: myObject.fval = 128.256;
18: myObject.uival = 512;
19: strncpy( myObject.strval, "This is a test.\n",
MAX_LINE );
20:
21: /* Send the message to the queue defined by the queue
ID */
22: ret = msgsnd( qid, (struct msgbuf *)&myObject,
23: sizeof(MY_TYPE_T), 0 );
24:
25: if (ret != -1) {
26:
27: printf( "Message successfully sent to queue %d\n",
qid );
28:
29: }
30:
31: }
32:
33: return 0;
34: }
```

Figure 6: Send message (msgsend.c)

Figure 7 shows an example to receive message from another process.

Q13. Correct the code.
Q14. Explain the output.

```
1: #include <stdio.h>
2: #include <sys/msg.h>
3: #define MAX_LINE 80
4:
5: #define MY_MQ_ID 111

6: typedef struct {
7: long type; // Msg Type (> 0)
8: float fval; // User Message
9: unsigned int uival; // User Message
10: char strval[MAX_LINE+1]; // User Message
11: } MY_TYPE_T;

12: int main()
13: {
14: MY_TYPE_T myObject;
15: int qid, ret;
16:
17: qid = msgget( MY_MQ_ID, 0 );
18:
```

```
19: if (qid >= 0) {
20:
21: ret = msgrcv( qid, (struct msgbuf *)&myObject,
22: sizeof(MY_TYPE_T), 1, 0 );
23:
24: if (ret != -1) {
25:
26: printf( "Message Type: %ld\n", myObject.type );
27: printf( "Float Value: %f\n", myObject.fval );
28: printf( "Uint Value: %d\n", myObject.uival );
29: printf( "String Value: %s\n", myObject.strval );
30:
31: }
32:
33: }
34:
35: return 0;
36: }
```

Figure 7: Receive message (msgreceive.c)

===XXX===