LAB 11

Processes

1. Creating a new process:

```
/*
 * proc1.c - creates a new process.
 */
#include <stdio.h>
#include <stdlib.h>

int main(void) {
   printf("Hi.\n");
   fork();
   printf("Bye.\n");
   exit(0);
}
```

2. Each process is running the same program:

```
/*
 * proc2.c - both parent and child processes run the same program.
 */
#include <stdio.h>
#include <stdlib.h>

#define N 10

int main()
{
    pid_t pid;
    char *message;
    int i;

    printf("fork program starting\n");
    pid = fork();
    switch(pid)
    {
      case -1:
        exit(1);
      case 0:
        message = "This is the child";
        break;
      default:
        message = "This is the parent";
        break;
    }

    for (i = 0; i < N; i++)
        puts(message);
    exit(0);
}
```

3. Each process is running a different program:

```
/*
 * proc3.c - both parent and child processes run different programs
 */

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

int main(void)
{
    pid_t pid;

    printf("fork program starting\n");
    pid = fork();
    switch(pid)
    {
     case -1:
        exit(1);

     case 0:
        execl("child", "child", (char *) 0);
        exit(1);     /* should never get here */

     default:
        printf("Process[%d]: parent in execution ...\n", getpid());
        sleep(2);
        if (wait(NULL) > 0) /* waiting for child */
           printf("Process[%d]: parent terminating child ...\n", getpid());
        printf("Process[%d]: parent terminating ...\n", getpid());
        exit(0);
    }
}


/*
 * child.c - the child program. This program replaces the parent's program.
 */
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    printf("Process[%d]: child in execution ...\n", getpid());
    sleep(1);
    printf("Process[%d]: child in execution ...\n", getpid());
    exit(0);
}
```

4. Writing your own pipe:

```
/*
 *  whowc.c - A poor man's "who | wc"
 */

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

int main(void)
{
   int fd[2];

   if (pipe(fd) == -1) {
         perror("Pipe");
         exit(1);
   }

   switch (fork()) {
   case -1:
         perror("Fork");
         exit(2);
   case 0:                          /* In the child */
         dup2(fd[1], STDOUT_FILENO);
         close(fd[0]);
         close(fd[1]);
         execl("/usr/bin/who", "who", (char *) 0);
         exit(3);
   default:                         /* In the parent */
         dup2(fd[0], STDIN_FILENO);
         close(fd[0]);
         close(fd[1]);
         execl("/usr/bin/wc", "wc", (char *) 0);
         exit(4);
   }
}
```

Exercise. Generalize **whowc.c** program: write a C program called **mypipe.c** that will take an arbitrary number of Unix commands at the command line and mimic Unix pipes as follows:

```
$ mypipe cmd1 cmd2 … cmdn
```

should do exactly the same operation as

```
$ cmd1 | cmd2 | … | cmdn
```

Recall that you may pass any number of strings as command-line arguments to your program using the following construct:

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

Here, **argc** is the number command-line arguments, and **argv** is an array of strings of commands including the program name itself.

To compile your C program, type

```
$ gcc –o mypipe mypipe.c
```

To run your program, type

```
$ ./mypipe command-line arguments
```