ITT440 – NETWORK PROGRAMMING

Introduction To Unix Multi-Process Programming

Basic Knowledge

- **Child Process Termination**
  - Once we have created a child process
    - parent process exits before the child
    - child exits before the parent
  - The wait() System Call
    - The simple way of a process to acknowledge the death of a child process
  - Asynchronous Child Death Notification
    - When a child process dies, a signal, SIGCHLD (or SIGCLD) is sent to its parent process.

- **Communications Via Pipes**
  - One of the mechanisms that allow related-processes to communicate
  - anonymous pipe.
  - one-way mechanism that allows two related processes (i.e. one is an ancestor of the other) to send a byte stream from one of them to the other one.
  - **The pipe() System Call**
    - used to create a read-write pipe that may later be used to communicate with a process we'll fork off
    - The call takes as an argument an array of 2 integers that will be used to save the two file descriptors used to access the pipe
    - first to read from the pipe, and the second to write to the pipe

Unix-fork-child.c

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

int main(void)
{
    pid_t pid;
    int rv;

    switch(pid = fork()) {
    case -1:
        perror("fork");  /* something went wrong */
        exit(1);         /* parent exits */

    case 0:
        printf(" CHILD: This is the child process!\n");
        printf(" CHILD: My PID is %d\n", getpid());
        printf(" CHILD: My parent's PID is %d\n", getppid());
        printf(" CHILD: Enter my exit status (make it small): ");
        scanf(" %d", &rv);
        printf(" CHILD: I'm outta here!\n");
        exit(rv);

    default:
        printf("PARENT: This is the parent process!\n");
        printf("PARENT: My PID is %d\n", getpid());
        printf("PARENT: My child's PID is %d\n", pid);
        printf("PARENT: I'm now waiting for my child to exit()...\n");
        wait(&rv);
        printf("PARENT: My child's exit status is: %d\n", WEXITSTATUS(rv));
        printf("PARENT: I'm outta here!\n");
    }

    return 0;
}
```

Compile the program above and note the output.

Exercise

- Write a program in C that fork() 2 childs and wait for the 2 childs to exit, and then the program will output the exit status of both child.

- Write a program in C that fork() 2 child but only wait for the 1 child to exit, and then the program will output the exit status of the waited child.

Unix-pipe.c

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

int main(void)
{
    int pfds[2];
    char buf[30];

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

    printf("writing to file descriptor #%d\n", pfds[1]);
    write(pfds[1], "test", 5);
    printf("reading from file descriptor #%d\n", pfds[0]);
    read(pfds[0], buf, 5);
    printf("read \"%s\"\n", buf);

    return 0;
}
```

Compile the program above and note the output.

Unix-fork_pipe.c

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

int main(void)
{
    int pfds[2];
    char buf[30];

    pipe(pfds);

    if (!fork()) {
        printf(" CHILD: writing to the pipe\n");
        write(pfds[1], "test", 5);
        printf(" CHILD: exiting\n");
        exit(0);
    } else {
        printf("PARENT: reading from pipe\n");
        read(pfds[0], buf, 5);
        printf("PARENT: read \"%s\"\n", buf);
        wait(NULL);
    }

    return 0;
}
```

Compile the program above and note the output.

Exercise

- Write a file in C that will fork 2 child. In the child, users are able to enter strings to write into the pipe. Parent program then will output the strings entered by users on both child. (HINT: you need to create two pipes).

Source

http://neuron-ai.tuke.sk/hudecm/Tutorials/C/special/multi-process/multi-process.html