Programmation Système
Contrôle Continu

Master 1 Informatique

12 Décembre 2011

2 hours – Papers and lecture notes permitted.

1 Comprehension questions (6 points)

Answer the questions concisely and accurately while justifying your answers. A simple yes or no without any explanation is not valid. Each question is worth 0.5 points with the exception of questions 4 and 5 which are each worth 1 point.

1. Justify the need for a « kernel » mode and a « user »mode.
2. In which file can you find the association between a user id and its user name (login) ?
3. Name one UNIX command which uses information from */etc/group* ? How is this information used ?
4. Why and how should we handle interrupted reads and writes ?
5. What do the flag *set-user-ID* on an executable file and the flag *sticky bit* on a directory mean ? Give an example of use for each flag.
6. In the process table is there only unfinished processes ? Why ?
7. Quote two information which are not the same for a parent process and its child.
8. The system call *wait* takes an integer as parameter which will be updated with the status of the child process which is waited. Does this integer contain only the return value associated to the child ? If not, how to extract this return value ?
9. Does the instructions located after a call to *execve* are executed ? Why ?
10. Does the reaction associated to a signal in a process is kept on a child process ?

2 Exercise with fork (3 points)

How many processes are created by the following piece of C code when it is compiled and run ? Draw the process tree which is created.

```
int main(int argc, char** argv) {
    int foo = fork();
    if (!foo) {
        fork();
        fork();
    }
    fork();

    return EXIT_SUCCESS;
}
```

3 Exercise my_popen (7 points)

The purpose of this exercise is to implement the procedure *my_popen*. To do this you must use the manual page for my_popen available in appendix A in order to understand how it works. The function *main* below provides an example to use the method *my_popen* :

```
int main(int argc, char* argv[]) {
    int f_lecture, f_ecriture;
    int nb_r, nb_w;
    f_lecture = open("/etc/passwd", O_RDONLY);
    f_ecriture = my_popen("more", "w");
    if (f_ecriture < 0)
        exit(1);
    while (nb_r = read (f_lecture, buffer, 100)) != 0)
        nb_w = write(f_ecriture, buffer, 100);
    return EXIT_SUCCESS;
}
```

The previous code reads the content of file */etc/passwd* and sends it to the *more* command which takes (when no parameter is specified) the bytes which are written on the standard input to display them on the screen. As explained in the manual page, the fact that we use *my_popen* with option *w* implies that each byte which is written on *f_ecriture* is sent on the standard input associated to the command *more*. When *my_popen* is used with option *r*, each byte written on the standard output is accessible through the file descriptor returned by *my_popen*.

4 Signals exercise (4 points)

Write a program which captures the signal *SIGSEGV*, by using system calls for reliable signal handling, and causes a serie of segfaults. The handler associated to *SIGSEGV* will receive 10 signals *SIGSEGV* before to display the pid of the process dealing with the handler and to terminate with return code 0.

Hint *A segmentation fault may be caused by accessing to an unallocated space of an array.*

A Manual page for my_popen

```
NAME
my_popen - Pipe to or from a process

SYNOPSIS
#include <stdio.h>

int my_popen(const char *commande, const char mode);

DESCRIPTION
The my_popen() function spawns a new process (fork) and create a pipe before
to return with an appropriated return code (see "return value" below). The
shell command is executed in the process which has been created. As a pipe is
unidirectionnal by definition, the "mode" parameter must only be used to read
or a write, and not the both. The corresponding file will be opened in read-only
or write-only.
The "commande" parameter is a pointer to a null-terminated string containing a
shell command line. This command is passed to /bin/sh using the -c flag.
Interpretation, if any, is performed by the shell. The "mode" argument is a
string which must contain either the letter 'r' for reading or the letter 'w'
for writing.
The return value from my_popen is a file descriptor which is opened for reading
or writing according to the "mode" argument. Writing to this file descriptor
writes on the standard input of the command. The command's standard output is the
same as that of the  process that called my_popen(), unless this is altered by the
command itself. Conversely, reading from a "popened" file descriptor reads the
command's standard output, and the command's standard input is the same as that of
the process that called my_popen().

RETURN VALUE
The my_popen function returns:
  * -1 if the fork(2) or pipe(2) calls fail ;
  * -2 the shell command which is executed terminates with a return code different from 0 ;
  * a file descriptor (value > 0) when no error has been detected.
```