PROCESSES AND SIGNALS

[] 1. Write a program in C with the name tsig, which tests synchronization mechanisms and signals. Use the following system functions: fork(), wait(), signal() or sigaction() and kill(). The program should run the algorithm from the section 2 with additional modifications from the section 3.

[] 2. Processes' algorithms without signal handlers

The main (parent) process algorithm

1. Create NUM_CHILD child processes, where NUM_CHILD is defined in the program, use the fork() function. Insert one second delays between consecutive fork() calls.

2. Check whether each of the child processes has been correctly created. If not, print an appropriate message, send to all already created child processes SIGTERM signal and finish with the exit code 1.

3. Print a message about creation of all child processes.

4. Call in a loop the wait() function, until receiving from the system information that there are no more processes to be synchronized with the parent one. Print a message that there are no more child processes. In this loop do count child processes terminations and, at the very end of the parent process, print a message with the number of just received child processes exit codes.

The child process algorithm

1. Print process identifier of the parent process

2. Sleep for 10 seconds

3. Print a message about execution completion

At this stage check whether your program works correctly. Only if it works correctly, continue with the tasks from the next section.

[] 3. Some modifications related to signal handlers

In the parent process

a. force ignoring of all signals with the signal() (or sigaction()) but after that at once restore the default handler for SIGCHLD signal

b. set your own keyboard interrupt signal handler (symbol of this interrupt: SIGINT)

c. the handler should print out info about receiving the keyboard interrupt and set some mark (global variable) which will notify about the fact of keyboard interrupt occurance

c. modify the main program in the following way: between the two consequtive creations of new processes check the mark which may be set by the keyboard interrupt handler. If the mark is set the parent process should signal all just created processes with the SIGTERM and, instead of printing the message about creation, print out a message about interrupt of the creation process. After that, the process should continue with wait()'s loop as before.

e. at the very end of the main process, the old service handlers of all signals should be restored.

In the child process

a. set to ignore handling of the keyboard interrupt signal

b. set your own handler of the SIGTERM signal, which will only print a message of the termination of this process.

[] 4. Additional remarks

a. two versions of the program are expected to be implemented, without and with signals (without changes from the section 3 and with them). The code should be in one source file and the version of compilation should be drived by the definition or by the lack of definition of the WITH_SIGNALS preprocessor symbol.

b. each printed message should start with the process identifier in the form child[PID] and in case of the parent process, parent[PID], e.g.:

parent[123]: sending SIGTERM signal

child[125]: received SIGTERM signal, terminating

[] 5. Hints

a. look at the following manual pages: fork(), wait(), signal(), sigaction(), kill(), getpid(), getppid(), sleep()

b. look at the content of the <signal.h> file

c. use for complex signal handling settings the symbol NSIG defined in standard include files

d. use stty command to get to know which keypress generates an interrupt from the keyboard (SIGINT) under your current shell configuration