LearningDiary_Test Diary

/* By Yunfeng He; 2012.11.13; */

In folder tests, there's Makefile, make to generate executables and make clean to clean

In my masktest program:

- first, the signal() function is used to install signal handler, of course, this is the most basic thing and take effect only once if there's no re-call in the handler
- then, sigprocmask() is tested. through my test, it shows that sigfillset and sigprocmask(SIG_BLOCK, ,);combined block any possible signal(apart form sigkill and sigterm).to unblock certain signal, just sigaddset certain signal and use sigprocmask(SIG_UNBLOCK,,) to unblock it.
- the test was: the parent process send two signals, if the sigset is emptyed, and then Unblock would take no effect, program exit after parent finish sending. if the sigset is emptyed and sigadd one sigint, then sigint is caught. if sigint and sigusr1 are all unblocked, then both signal can be caught.

Sigwaittest program:

- another test is about sigwait(). the pause() function would wait for any signal and when signal arrives, certain handler will be called. this means for a signal that I do not want to "handle" but need to notice, I will have to call a do nothing handler(which means calling function and waste sys resources), just to proceed(jump out of pause state). sigwait() can be used to replace pause().
- sigwait() must be used as such: the signal that we wanna wait as a "go" signal must be put in our sigset, and this set shall be blocked!. even it's blocked, sigwait()can detect it and write the number of the pending signal to the second argument(which shall not be NULL).
- sigwait() suspend the thread, meaning that other signals will also take no effect(??)
- -lpthread shall be used to compile since sigwait() has to include <pthread.h>
- my sigwaittest program will create an orphan there.(sloved by send SIGKILL before parent return.)
- program works fine at first several tries, keep launching it many times and it blocks. This is due to unreliable handler design. sigwait() is a bad idea anyway, may cause the

process to block forever if signal comes just before sigwait() is called.

- The solution is to write better handler and used better mechanism, the signalbetter example works fine!!!

IPCpipe program:

- pipe is a "stream" container that reside in the kernel, after reading from pipe, the "data" will no longer exist.
- This program is the practice of the signal better example. With fork() and two processes involved; two pipes are established and after fork(), each process would cut all ties with one pipe, only to use the pipe of itself to buffer signal.
- If a program wanna use pipe to communicate, keep in mind that two pipes are needed to make bidirectional communication; Because essentially, the pipe() function create a system level buffer, and after fork, both process can write and read from the same underlayer buffer, creating odd results.
- When using two pipes for bidirectional communication, do remember to close unused end of the pipe at each process.
- It proves that the mechanism work perfect, without possibility of block. Because the Mechanism of this signal handling is that the signal handler use pipe's internal buffer to buffer all coming signals, signals are read only when it's needed. In this way, the program can do whatever it is doing and don't have to care about when a signal would come.
- This test program lies the foundation for my morsetranscoder program!!