Responsible lecturer: Illés, Zoltán associate professor
Media & Educational Informatics Department

Pipes

Goal: We are getting to know the usage of unnamed and named pipes. A pipe is a FIFO data structure. We are going to get acquainted with the possibility to wait data from several pipes in the same time.

We learn: *pipe – to create an unnamed pipe (include unistd, sys/stat.h); mkfifo – to create a named pipe (include unistd, sys/stat.h); read, write, close – operations with unnamed and named pipes; open unlink – operations with named pipes; poll, ppoll – wait data from several pipes (generally from any data structures given by file descriptors) (include poll.h); pollfd – struct for polling; POLLIN, POLLOUT – constants for polling input or output; select – wait data from several pipes (include signal.h, sys/select.h); FD_SET, FD_ZERO, FD_ISSET – operations using select*

Tasks

1. Write a C program with a child. The parent should write a „Hello" into the unnamed pipe and the child has to read them out! The parent waits for the end of child as usual! (*Remember, a read waits for a data from the pipe – if there is no data it waits forever! Try it by using sleeps! Usually we close the not used end of the pipe!)*

```
int pipe(int pipefd[2]);
//pipefd – file descriptors of the file
//pipfd[0] – reading end, pipefd[1] – writing end
```

2. Modify the program – the children should write back a „Hello parent" using an unnamed pipe. *(Firstly try it with two different pipes – one for sending a message to the child and the other to send a message to the parent. It is good but not really nice. May we use the same pipe for the whole conversation? Yes, but be careful! You have to avoid to read back your own message from the pipe! Use some synchronozation e.g. send a signal to the other process that you finished writing!)*

3. Write a chat program which uses a named pipe for sending messages! You can start it twice from different telnet windows to be able to have a chat with „yourself"! (*There is the same problem as before – you have to avoid reading back your own message! Use one of the above mentioned solutions! You have to now that read waits forever a new data from the pipe except the writing end is closed. Be careful, if there are several writing ends in several processes each has to be closed!)*

4. Write a C program with two child processes and two pipes (one for each). The children have to create random numbers between 1 and 50 (the first child) and between 51 and 100 (the second child). They write the just created number to their own pipe after it they wait for some random time before the next one. The parent has to read the arriving numbers from the pipes! Please, use poll/ppoll for the solution! *(If you used two read for the two pipes it would not be good! Remember, a read waits for an arriving data from the given pipe and do not let the parent to examine the other pipe whether there is a data there or not!)*

```
int poll(struct pollfd *fds, nfds_t nfds, int timeout);
// fds – you can see below the structure, which events you want to watch
// nfds – number of file descriptors you watch
int ppoll(struct pollfd *fds, nfds_t nfds,
                const struct timespec *tmo_p, const sigset_t *sigmask);
//it has a timer and a signal mask as well

struct pollfd {
                int   fd;         /* file descriptor */
                short events;     /* requested events */
                short revents;    /* returned events */
            };
```

Financed from the financial support ELTE won from the Higher Education Restructuring Fund of the Hungarian Government

```
struct timespec {
               long    tv_sec;         /* seconds */
               long    tv_nsec;        /* nanoseconds */
           };
```

5. Modify the above written C program and use select for the same task. *(For select you have to give three group of file descriptors to input, output and exceptions. Remember, after select call you have to give again the sets!)*

```
int select(int nfds, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout);
//nfds – the greatest file descriptor, the watchable readings, writings,
exceptions and a timer)

void FD_CLR(int fd, fd_set *set);
//initialize the set
int  FD_ISSET(int fd, fd_set *set);
void FD_SET(int fd, fd_set *set);
//to set the set
void FD_ZERO(fd_set *set);
//clear the set
```

6. Modify the your chat program using poll or select!