CSI 402 – Systems Programming – Handout 15.4
An Example for `select` System Call

Note: This example, taken from pages 166–167 of the text by Haviland et al., shows how `select` system call can be used by a parent process to monitor the read-ends of pipes with three children and also `stdin`.

The parent process creates three pipes and three children. Each child sends messages to the parent through a pipe. The parent then waits in a loop, using the `select` system call to check the read-ends of the three pipes as well as `stdin`. Each child writes two messages and then exits. The parent waits for all the children to exit. When this program is executed, the messages sent by children and the characters typed by a user from the keyboard will all be interleaved.

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/time.h>
#include <sys/wait.h>

#define  MSGSIZE  6
#define  NPIPES   3
#define  NDESC    2

int  parent (int [] []);
int  child (int []);
char *msg1 = "hello";   char *msg2 = "bye!!";

int main(void) {
  int  pfd[NPIPES][NDESC];   /* To get file descriptors for three pipes. */
  int  i;                    /* Temporary. */

  /* Set up NPIPES pipes and fork as many children. */
  for (i = 0; i < NPIPES; i++) {
    if (pipe(pfd[i]) == -1) {
       fprintf(stderr, "Call to pipe failed.\n"); exit(1);
    }

    switch (fork()) {
      case (pid_t) -1: /* Fork failed. */
                       fprintf(stderr, "Call to fork failed.\n"); exit(1);

      case  0:  /* Child process. */
                child(pfd[i]);

    } /* End of switch. */
  }
  parent(pfd);  /* Parent will monitor all the three pipes and stdin. */
  return 0;
} /* End of main. */
```

```
int parent (int p[NPIPES][NDESC]) {

  /* Code for parent process (server). */
  char buf[MSGSIZE], tc;
  fd_set  tset, master;   /* Data structure needed for select. */
  int  nfd;   /* No. of file descriptors of interest. */
  int  i;

  /* The parent doesn't need the file descriptors for the write-ends of pipes. */
  /* So, they are closed.                                                       */
  for (i = 0; i < NPIPES; i++) {
    if (close(p[i][1]) == -1) { /* Failed to close write end of pipe. */
       fprintf(stderr, "Parent: Couldn't close write end of pipe %d.\n", i);
       exit(1);
    }
  }

  /* Set the bit masks for the select system call. */
  FD_ZERO(&master);
  FD_SET(STDIN_FILENO, &master);  /* Parent wants to watch stdin and  */
  for (i = 0; i < NPIPES; i++)    /* the read-ends of three pipes.    */
    FD_SET(p[i][0], &master);

  /* Parent calls select without timeout. So, parent will block until */
  /* an event occurs.                                                 */

 /* We need to remember the master bit mask since select will change the mask. */
  tset = master;
  nfd  = p[NPIPES-1][0]+1;  /* The no. of file descriptors must include stdin. */
  while (select(nfd, &tset, NULL, NULL, NULL) > 0) {

    /* Check if stdin has any data and print it. */
    if (FD_ISSET(STDIN_FILENO, &tset)) {
       printf("From stdin: "); read(STDIN_FILENO, &tc,1);
       printf("%c\n", tc);
    }

    /* Check if any of the pipes has data and print the data. */
    for (i = 0; i < NPIPES; i++){
       if (FD_ISSET(p[i][0], &tset)) {
         if (read(p[i][0], buf, MSGSIZE) > 0)
            printf("Message from Child %d: %s\n", i, buf);
       }
    } /* End of for. */

    /* The server will return to the main program if all children have exited. */
    if (waitpid(-1, NULL, WNOHANG) == -1)
       return 0;
```

```
    /* Otherwise, we need to restore the master bit mask and call select again. */
    tset =  master;
  } /* End of while. */
  return 0;
} /* End of parent. */


int child (int p[]) {

  #define  NUM_MSG  2
  #define  DIVISOR  4

  /* Code for each child process. Each child process writes two messages */
  /* with a different waiting time between the messages and quits.       */
  int  count;

  /* Close the read-end of the pipe. */
  if (close(p[0]) == -1) { /* Failed to close read end of pipe. */
     fprintf(stderr, "Child: Couldn't close read end of pipe.\n");
     exit(1);
  }

  /* Send messages through the pipe. */

  for (count = 0; count < NUM_MSG; count++) {
     write(p[1], msg1, MSGSIZE);
     sleep(getpid() % DIVISOR);
  }
  /* Send the final message. */
  write(p[1], msg2, MSGSIZE);
  exit(0);

} /* End of child. */
```

Sample output: After starting the program, the user typed the character `'x'` followed by `'\n'`. Notice that both of these characters are written to the output along with the messages from Children 0, 1 and 2.

```
Message from Child 0: hello
Message from Child 1: hello
Message from Child 2: hello
Message from Child 0: hello
From stdin: x
From stdin:

Message from Child 1: hello
Message from Child 0: bye!!
Message from Child 2: hello
Message from Child 1: bye!!
Message from Child 2: bye!!
```