Seminar 4: Communication
24296-Sistemes Operatius

1 Shell Commands

Write a program (txt2bin) reads numbers from the standard input in text format separated by spaces and converts them into binary format and writes them to the standard output. For the conversion from text to int you can use functions sscanf or atoi. For reading you can use read_split.

Then write a program (bin2txt) that reads a binary file which contains int numbers one after the other and generates in the standard output the same numbers in text format separated by spaces. For translating you can use the function sprintf which stores in the buffer the number given as int:

```
int num, fd;
char ch_end, buf[80];
sscanf(buff, "%d", &num)                    // conversion from text to int
sprintf(buff, "%d", num)                    // conversion from int to text
read_split(fd, buff, 80, &ch_end)
```

2 Pipes

Although it is not usual, standard input and output can be used to exchange binary data. Assuming you have implemented and compiled the two precedent shell commands (called txt2bin bin2txt), you can now use the shell command lines below to write the same text list of numbers (separated by spaces) from the standard input to the standard output:

```
> txt2bin | bin2txt                          // blocks for keyboard input
> echo "12 18 33" | txt2bin | bin2txt       // sends the string to std out
```

Then implement the second part of the precedents shell commands (txt2bin pipe bin2txt) using a single program that will create one or two processes, using a pipe and the sys-call dup2. Here follow some examples of sys-call usages:

```
int fd[2];
pipe(fd)
dup2(fd[0],0)    // replace standard in by pipe read
dup2(fd[1],1)    // replace standard out by pipe write
```

3 Sockets

Program a server that accepts client connections and reads 100 numbers from each client and maintains a sum of the total quantity. Program also the client that sends numbers to the server. Here you have the main operations of both Server and client:

```
void Server(){
  sockfd = socket(AF_INET, SOCK_STREAM, 0);
  fill_addr(NULL, argv[1], &serv_addr);
  bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr));
  listen(sockfd,100);
  newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);
  // receive from client via newsockfd
}

void Client(){
  sockfd = socket(AF_INET, SOCK_STREAM, 0);
  fill_addr(argv[1], argv[2], &serv_addr);
  connect(sockfd,(struct sockaddr *) &serv_addr, sizeof(serv_addr));
  // Send to server via sockfd
}
```

4 Collaborative sum with Shared Memory

Implement two separate shell commands that each ask for a common pointer to an integer using shared memory key 1234. They then implement a collaborative sum correctly synchronized with named semaphores. Here helper declarations and signatures:

```
#include <semaphore.h>
int shmkey = 1234;
shmid = shmget(shmkey, SHM_SIZE, 0666 | IPC_CREAT);
int *shmpointer = shmat(shmid, NULL, 0);
sem_t* named_mutex;
sem_open("named_mutex", O_CREAT, 0600, 1);  // init to 1 for a mutex
sem_wait(named_mutex);
sem_post(named_mutex);       // equivalent to sem_signal
sem_close(named_mutex);      // very important; if not reusing may not work
```

5 A Distributed Shell with Sockets

Implement a multi process application in which a process receives client connections (accept) asking for a shell command to be executed (sent by text through the socket connection). Then the process will create a child process serving the petition.

6 Delivery exercises

1. Implement a program that creates 10 child processes and each sends 10 integer numbers to the same pipe. The father then takes all the numbers received in the pipe and sums them together and prints the result.

2. Implement a collaborative multiplication using shared memory. Two different processes are going to ask for a shared memory segment to share an array of 100 integers. One process is going to initialize it to 1. When ready both are going to multiply each entry by its corresponding index.

At the end the array at entry i should be $A[i] = i * i$. You will need to use named semaphores.