Sistemas de Operação (2018/2019)
Ficha 6

Q1. Considere o seguinte programa que implementa uma *pipe* entre processos pai e filho. Compile o exemplo e execute-o. Leia o código com atenção e compreenda-o.

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

#define PP_RD 0
#define PP_WR 1

int main(int argc, char* argv[]) {
  int   n, r, fd[2];
  pid_t pid;
  char  line[BUFSIZ];

  if (pipe(fd) < 0) {
    perror("pipe error");
    exit(EXIT_FAILURE);
  }

  if ((pid = fork()) < 0) {
    perror("fork error");
    exit(EXIT_FAILURE);
  }
  else if (pid > 0) {
    /* parent */
    close(fd[PP_RD]);
    printf("P=> Parent process with pid %d\n", getpid());
    printf("P=> Messaging the child process (pid %d):\n", pid);
    snprintf(line, BUFSIZ, "Hello! I'm your parent pid %d!\n", getpid());
    if ((r = write(fd[PP_WR], line, strlen(line))) < 0) {
      fprintf(stderr, "Unable to write to pipe: %s\n", strerror(errno));
    }
    close(fd[PP_WR]);
    /* wait for child and exit */
    if ( waitpid(pid, NULL, 0) < 0) {
```

```
      fprintf(stderr, "Cannot wait for child: %s\n", strerror(errno));
    }
    exit(EXIT_SUCCESS);
  }
  else {
    /* child */
    close(fd[PP_WR]);
    printf("C=> Child process with pid %d\n", getpid());
    printf("C=> Receiving message from parent (pid %d):\n", getppid());
    if ((n = read(fd[PP_RD], line, BUFSIZ)) < 0 ) {
      fprintf(stderr, "Unable to read from pipe: %s\n", strerror(errno));
    }
    close(fd[PP_RD]);
    /* write message from parent */
    write(STDOUT_FILENO, line, n);
    /* exit gracefully */
    exit(EXIT_SUCCESS);
  }
}
```

Altere o programa de tal forma que, em vez das mensagens enviadas, o processo pai abra um ficheiro de texto (cujo nome é dado na linha de comando), leia o seu conteúdo e o passe através da pipe para o processo filho. Este deverá receber o conteúdo do ficheiro e escrevê-lo no stdout. Compile e execute o seu programa com um ficheiro de texto grande (p.e., o ficheiro com este código fonte).

Q2. O programa seguinte implementa um mecanismo de comunicação semelhante entre processos pai e filho, usando um par de *sockets*. Ao invés das pipes, os sockets permitem a comunicação bidirecional. Compile e execute o programa. Leia com atenção o código e compreenda-o.

```
#include <sys/wait.h>
#include <sys/socket.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

#define SOCK0 0
#define SOCK1 1

#define DATA0 "In every walk with nature..."
#define DATA1 "...one receives far more than he seeks."
/* by John Muir */
```

```
int main(int argc, char* argv[]) {
  int   sockets[2];
  char  buf[1024];
  pid_t pid;

  if (socketpair(AF_UNIX, SOCK_STREAM, 0, sockets) < 0) {
    perror("opening stream socket pair");
    exit(1);
  }

  if ((pid = fork()) < 0) {
    perror("fork");
    return EXIT_FAILURE;
  }
  else if (pid == 0) {
    /* this is the child */
    close(sockets[SOCK0]);
    if (read(sockets[SOCK1], buf, sizeof(buf)) < 0)
      perror("reading stream message");
    printf("message from %d-->%s\n", getppid(), buf);
    if (write(sockets[SOCK1], DATA1, sizeof(DATA1)) < 0)
      perror("writing stream message");
    close(sockets[SOCK1]);
    /* leave gracefully */
    return EXIT_SUCCESS;
  }
  else {
    /* this is the parent */
    close(sockets[SOCK1]);
    if (write(sockets[SOCK0], DATA0, sizeof(DATA0)) < 0)
      perror("writing stream message");
    if (read(sockets[SOCK0], buf, sizeof(buf)) < 0)
      perror("reading stream message");
    printf("message from %d-->%s\n", pid, buf);
    close(sockets[SOCK0]);
    /* wait for child and exit */
    if (waitpid(pid, NULL, 0) < 0) {
      perror("did not catch child exiting");
      return EXIT_FAILURE;
    }
    return EXIT_SUCCESS;
  }
```

```
}
```

Que tipo de socket está a ser usado e qual o seu âmbito de utilização? Modifique o programa de tal forma que o processo pai abra um ficheiro de texto e transfira o seu conteúdo para o processo filho. Por sua vez o processo filho deve receber o conteúdo, passar todos os caracteres para maiúsculas e devolvê-los para o processo pai que os imprime no stdout.

Q3. Considere agora o seguinte programa que implementa um servidor que recebe mensagens de clientes e as imprime no stdout. Esta implementação utiliza sockets `AF_INET` o que significa que o servidor (e os clientes) podem localizar-se em máquinas distintas numa rede de computadores. Leia o código e compreenda-o.

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main(int argc, char* argv[]) {
  int       sock;
  socklen_t length;
  struct sockaddr_in name;
  char buf[1024];

  /* create socket from which to read */

  if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
    perror("opening datagram socket");
    return EXIT_FAILURE;
  }

  /* create name with wildcards - port will be given by OS */
  name.sin_family = AF_INET;
  name.sin_addr.s_addr = INADDR_ANY;
  name.sin_port = 0;
  if (bind(sock, (struct sockaddr *)&name, sizeof(name))) {
    perror("binding datagram socket");
    return EXIT_FAILURE;
  }

  /* find assigned port value and print it out */
  length = (socklen_t)sizeof(name);
```

```
  if (getsockname(sock, (struct sockaddr *)&name, &length)) {
    perror("getting socket name");
    return EXIT_FAILURE;
  }
  printf("Socket has port #%d\n", ntohs(name.sin_port));

  /* read from the socket */
  if (read(sock, buf, sizeof(buf)) < 0)
    perror("receiving datagram packet");
  printf("received:\n%s\n", buf);
  close(sock);
  return EXIT_SUCCESS;
}
```

Compile o programa (`gcc -Wall server.c -o server`) e execute-o fazendo `./server &`. O servidor fica a executar em *background*, desacoplado da shell e à escuta de pedidos na porta por ele indicada.

Q4. O programa seguinte mostra um cliente para o servidor anterior. Leia o código e compreenda-o.

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>

#define DATA "Such is the nature of evil. \
Out there in the vast ignorance of the world it \
festers and spreads. A shadow that grows in the dark. \
A sleepless malice as black as the oncoming wall of night. \
So it ever was. So will it always be. \
In time all foul things come forth."

/*
 * this is the client side
 */
int main(int argc, char *argv[]) {
  int    sock;
  struct sockaddr_in name;
```

```
  struct hostent *hp;

  /* create socket on which to send */
  if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
    perror("opening datagram socket");
    return EXIT_FAILURE;
  }

  /* destination is constructed from hostname
     and port both given in the command line */
  if ((hp = gethostbyname(argv[1])) == 0) {
    fprintf(stderr, "%s: unknown host\n", argv[1]);
    return EXIT_FAILURE;
  }
  memcpy(&name.sin_addr, hp->h_addr, hp->h_length);
  name.sin_family = AF_INET;
  name.sin_port = htons(atoi(argv[2]));

  /* connect to given port */
  if(connect(sock, (struct sockaddr *)&name, sizeof(name)) < 0) {
    perror("connecting to server socket");
    return EXIT_FAILURE;
  }

  /* send message */
  if (write(sock, DATA, sizeof(DATA)) < 0)
    perror("sending datagram message");

  close(sock);

  return EXIT_SUCCESS;
}
```

Note que recebe 2 argumentos: o nome/endereço da máquina e o número da porta em que o servidor escuta. Estes dois argumentos servem para identificar sem ambiguidade o processo servidor em qualquer localização na rede. Compile o programa e execute-o fazendo `./client localhost port` em que `port` é o número da porta indicado pelo servidor. O que acontece quando usa um número de porta diferente?
Note que o servidor só atende o primeiro cliente. Altere o programa para que o servidor esteja sempre disponível. Altere o servidor por forma a que use uma porta fixa. O que tem de alterar no cliente? E na forma de executá-lo?

Q5. Altere os programas do servidor e do cliente por forma a implementar um serviço de citações. Neste serviço os clientes enviam um pedido ao servidor com a marca `GET`. O servidor responde com uma citação aleatória (uma string) de uma colecção que mantém num array. As mensagens enviadas têm a forma `QUOTE:string`. Compile e teste o programa. Altere o programa para que o servidor receba pedidos `GET:n`, em que `n` é um inteiro que indica o número de citações que devem ser retornadas pelo servidor para o cliente.
O idioma utilizado na conversa entre o servidor e o cliente é designado por *protocolo*.

Q6. Partindo do mesmo código cliente-servidor, implemente um serviço simples de SMS. A interacção entre os clientes e o servidor dá-se de acordo com o seguinte protocolo:

- um cliente faz `PUT:username:message` para enviar um SMS destinado ao utilizador `username` para o servidor. O servidor guarda as mensagens recebidas numa estrutura de dados apropriada, e.g., uma lista.

- um cliente faz `GET:username` para receber do servidor todas as mensagens aí guardadas destinadas a`username`. O servidor envia primeiro ao cliente uma mensagem da forma `INCOMMING:n`, indicando o número de mensagens que vai enviar. Seguidamente envia os SMS, um a um, em mensagens da forma `SMS:message`. Depois de enviadas as mensagens pendentes para um cliente, o servidor descarta-as.