Sistemas de Operação (2018/2019)
Ficha 7

Q1. Exemplo mostra como ligar o stdout do comando `cmd1` ao stdin do comando `cmd2`, usando uma "pipe". Analise o código, complete-o, compile-o e execute-o.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

char* cmd1[] = {"ls", "-l", NULL};
char* cmd2[] = {"wc", "-l", NULL};

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

  if (pipe(fd) < 0) {
    /* pipe error */
  }

  if ((pid = fork()) < 0) {
    /* fork error */
  }

  if (pid > 0) {
    close(fd[0]);
    dup2(fd[1], STDOUT_FILENO);  // stdout to pipe
    close(fd[1]);
    // parent writes to the pipe
    if (execvp(cmd1[0], cmd1) < 0) {
      /* exec error */
    }
  } else {
    close(fd[1]);
    dup2(fd[0], STDIN_FILENO);  // stdin from pipe
    close(fd[0]);
    if (execvp(cmd2[0], cmd2) < 0) {
      /* exec error */
    }
  }
}
```

Q2. Use o código anterior para escrever um comando `piper` que recebe uma sequência de argumentos semelhante a uma "pipe" na "shell" (p.e., `piper "ls -l | wc -l"`), e que produza os "arrays" `cmd1` e `cmd2` e execute a "pipe".

Q3. O exemplo seguinte mostra a manipulação e o tratamento de sinais pelo utilizador. Na função `main`, aparece a função `signal` que regista qual o tratamento que deve ser dado, quando o processo que executa o código recebe os sinais `SIGUSR1` e `SIGUSR2`. Para testar o exemplo, abra um terminal novo envie o sinal `SIGUSR1` ao processo N usando o comando `kill -SIGUSR1 N`.

```
#include <errno.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

static void handler1() { printf("received SIGUSR1\n"); }

static void handler2() { printf("received SIGUSR2\n"); }

static void handler3() { printf("received SIGHUP\n"); }

int main(int argc, char* argv[]) {
  printf("My PID is %d\n", getpid());
  if (signal(SIGUSR1, handler1) == SIG_ERR) {
    fprintf(stderr, "Can't catch SIGUSR1: %s", strerror(errno));
    exit(EXIT_FAILURE);
  }
  if (signal(SIGUSR2, handler2) == SIG_ERR) {
    fprintf(stderr, "Can't catch SIGUSR2: %s", strerror(errno));
    exit(EXIT_FAILURE);
  }
  if (signal(SIGHUP, handler3) == SIG_ERR) {
    fprintf(stderr, "Can't catch SIGHUP: %s", strerror(errno));
    exit(EXIT_FAILURE);
  }

  /* stick around ... */
  for ( ; ; )
    pause();
}
```

Q4. Estenda o código anterior para que suporte o tratamento dos sinais `SIGTSTP` (enviado pelo terminal quando se usa `CTRL-Z`) e `SIGINT` (enviado pelo terminal quando se usa `CTRL-C`), imprimindo nesses casos uma mensagem adequada. Por último, estenda o código para que suporte o tratamento do sinal `SIGKILL`.

Q5. O exemplo seguinte mostra como pode fazer a troca de sinais entre um processo pai e um processo filho. Analise o código, complete-o, compile-o e execute-o.

```
#include <errno.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

static void handler_parent()
  { printf("%d: Parent received signal\n", getpid()); }
static void handler_child()
  { printf("%d: Child received signal\n", getpid()); }

int main(int argc, char* argv[]) {
  pid_t pid;
  if (signal(SIGUSR1, handler_parent) == SIG_ERR)
    { /* signal error */}
  if (signal(SIGUSR2, handler_child) == SIG_ERR)
    { /* signal error */}
  if ((pid = fork()) < 0)
    { /* fork error */}
  else if (pid > 0) {
    /* parent's code */
    kill(pid, SIGUSR2);
    pause();
  } else {
    /* child's code */
    kill(getppid(), SIGUSR1);
    pause();
  }
}
```

Q6. Estenda o código anterior de forma a que o processo filho envie 3 sinais ao processo pai. O processo pai, que não sabe o número de sinais que irá receber, deverá fazer uma contagem dos sinais que recebe e imprimir essa contagem cada vez que recebe um sinal.

Q7. O exemplo seguinte mostra a implementação de um alarme simples, usando um temporizador que quando chega a zero envia um sinal de alarme para o próprio processo e reinicia o temporizador. As características do alarme e temporizador podem ser definidas pelo utilizador, através da manipulação das respectivas estruturas de dados associadas ao sistema operativo. No terminal, use a função `man sigaction` e `setitimer`, para perceber melhor as potencialidades das funções. Analise o código, compile-o e execute-o.

```
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <sys/time.h>

void timer_handler (int signum) {
  static int count = 0;
  printf ("timer expired %d times\n", ++count);
}

int main (int argc, char* argv[]) {
  struct sigaction sa;
  struct itimerval timer;

  /* Install timer_handler as the signal handler for SIGVTALRM. */
  memset (&sa, 0, sizeof (sa));
  sa.sa_handler = &timer_handler;
  sigaction (SIGVTALRM, &sa, NULL);

  /* Configure the timer to expire after 250 msec... */
  timer.it_value.tv_sec = 0;
  timer.it_value.tv_usec = 250000;
  /* ... and every 250 msec after that. */
  timer.it_interval.tv_sec = 0;
  timer.it_interval.tv_usec = 250000;
  /* Start a virtual timer.
     It counts down whenever this process is executing. */
  setitimer (ITIMER_VIRTUAL, &timer, NULL);

  /* Do busy work. */
  for( ; ; )
    ;
}
```

Q8. Estenda o código anterior de forma a que o processo termine no final de 3 intervalos de temporização.