Felipe Cancella - 2210487
Diego Miranda - 2210996

EX1

Objetivo;
O objetivo do programa é anular a execução do comando ctrl+c, permitindo apenas que o programa seja interrompido através do comando ctrl+\ através do uso de sinais.

Estrutura do programa;
Função main - Espera para ler um sinal de interrupção, e chama a função respectiva para exibição de mensagem.

```
int main (void){

  void (*p)(int);// ponteiro para função que recebe int como parâmetro

  // signal intercepta quando le o comando SIGINT (Ctrl+c), e chama a funcao intHandler no lugar
  p = signal(SIGINT, intHandler);
  printf("Endereco do manipulador anterior %p\n", p);

    // signal intercepta o comando SIGQUIT (Ctrl+\), e executa a funcao quitHandler
  p = signal(SIGQUIT, quitHandler);
  printf("Endereco do manipulador anterior %p\n", p);
  printf("Ctrl-C desabilitado. Use Ctrl-\\ para terminar\n");

  for(EVER);
}
```

Função intHandler - Recebe um sinal, e exibe a mensagem.

```
void intHandler(int sinal){
  printf("Você pressionou Ctrl-C (%d)\n", sinal);
}
```

Função quitHandler - Recebe um sinal, exibe a mensagem, e termina a execução do programa.

```
void quitHandler(int sinal){
  printf("Terminando o processo...\n");
  exit (0);
}
```

Resultado:

```
Endereco do manipulador anterior (nil)
Endereco do manipulador anterior (nil)
Ctrl-C desabilitado. Use Ctrl-\ para terminar
```

^CVocê pressionou Ctrl-C (2)
^CVocê pressionou Ctrl-C (2)
^CVocê pressionou Ctrl-C (2)
^\Terminando o processo...

Solução;
Se mantermos o signal, então os comandos serão interrompidos, e a devida função será chamada como substituição. Assim, o comando SIGQUIT (ctrl+\ OU 'kill -s 3 num_pid') será responsável por fechar.
Quando o comando signal é removido, o código para de interceptar o sinal SIGINT (ctrl+c OU 'kill -s 2 num_pid'), e passa a executar ele normalmente. Com isso, ao digitar ctrl+c o comando não é interceptado, e o código é interrompido normalmente.

EX2

Objetivo;
O objetivo do programa é anular a execução do sinal SIGKILL, e analisar os resultados.

Estrutura do programa;
Função main - Espera para ler um sinal de interrupção, e chama a função respectiva para exibição de mensagem.

```
int main (void){

  void (*p)(int);// ponteiro para função que recebe int como parâmetro

  printf("Pid id: %d\n", getpid());

  printf("kill SIGKILL desabilitado\n");

  // Tenta interceptar o SIGKILL (kill -s KILL 'pid_id')
  p = signal(SIGKILL, intHandler);

  for(EVER);
}
```

Função intHandler - Recebe um sinal, e exibe a mensagem.

```
void intHandler(int sinal){
  printf("Você tentou dar SIGKILL %d\n", sinal);
}
```

Solução;

O comando SIGKILL (kill -9 'pid_id' OU kill -s KILL 'pid_id') não é interceptável, sendo assim, ao tentar identificar ele e substituir, nada acontece, e o código é na verdade interrompido.

EX3

Objetivo;
O objetivo do enunciado é usar o programa filhocidio.c utilizado em aula com quatro programas auxiliares diferentes, e analisar os resultados

A)
Estrutura do programa;
Função main -

```
int delay = 5; //Botando delay do pai manualmente

int main (int argc, char *argv[]){

  pid_t pid;
  signal(SIGCHLD, childhandler);
  if ((pid = fork()) < 0){
    fprintf(stderr, "Erro ao criar filho\n");
    exit(-1);
  }

  if (pid == 0) { //child
    for(EVER); // Filho roda infinitamente
  }

  else {// parent
    printf("Delay: %d\n", delay);
      sleep(delay); // delay que o pai espera a execucao do filho, esse delay substitui o papel do waitpid. Se passar do tempo, o pai mata o filho ao inves de esperar ele acabar
    printf("Program %s exceeded limit of %d seconds!\n", argv[2], delay);
    kill(pid, SIGKILL);
  }
  return 0;
}
```

Função childHandler - Espera filho ser encerrado, e exibe mensagem caso tenha sido antes do fim da execução do pai

```
void childhandler(int signo) // Executed if child dies before parent
{
  int status;
  pid_t pid = wait(&status);
    printf("Child %d terminated within %d seconds com estado %d.\n", pid, delay, status);
  exit(0);
```

```
}
```

Solução;
O Código do pai interrompe o processo do filho, já que o filho roda infinitamente, e o pai tem delay máximo de 3.

B)
Estrutura do programa;
Função main -

```
int delay = 5; //Botando delay do pai manualmente

int main (int argc, char *argv[]){

  pid_t pid;

  signal(SIGCHLD, childhandler);

  if ((pid = fork()) < 0){
    fprintf(stderr, "Erro ao criar filho\n");
    exit(-1);
  }

  if (pid == 0) { // child
    sleep(3); // Filho dorme por 3 segundos e termina
    exit(0);
  }

  else {// parent

    printf("Delay: %d\n", delay);
      sleep(delay); // delay que o pai espera a execucao do filho, esse delay substitui o papel do waitpid. Se passar do tempo, o pai mata o filho ao inves de esperar ele acabar
    printf("Program %s exceeded limit of %d seconds!\n", argv[2], delay);
    kill(pid, SIGKILL);
  }
  return 0;
}
```

Função childHandler - Espera filho ser encerrado, e exibe mensagem caso tenha sido antes do fim da execução do pai

```
void childhandler(int signo) // Executed if child dies before parent
{
  int status;
  pid_t pid = wait(&status);
```

```
   printf("Child %d terminated within %d seconds com estado %d.\n", pid, delay, status);
  exit(0);
}
```

Solução;
O código do filho roda por 3 segundos, e como o delay do pai era de 5s, então o código vai finalizar sinalizando que o comando filho acabou dentro do limite.

C)
Estrutura do programa;
Função main -

```
int delay = 7; //Botando delay do pai manualmente

int main (int argc, char *argv[]){

  pid_t pid;

   signal(SIGCHLD, childhandler); //sinal que o filho passa pro pai quando termina

  if ((pid = fork()) < 0){
   printf("Erro ao criar filho\n");
   exit(-1);
  }

  if (pid == 0) { //child
   printf("%s\n", argv[1]);
   execvp(argv[1], argv); // Passamos pro execvp a funcao sleep5
  }

  else { // parent

   printf("Delay: %d\n", delay);
     sleep(delay); // delay que o pai espera a execucao do filho, esse delay substitui o papel do waitpid. Se passar do tempo, o pai mata o filho ao inves de esperar ele acabar
   printf("Program %s exceeded limit of %d seconds!\n", argv[1], delay);
   kill(pid, SIGKILL);
  }
  return 0;
}
```

Função childHandler - Espera filho ser encerrado, e exibe mensagem caso tenha sido antes do fim da execução do pai

```
void childhandler(int signo) // Executed if child dies before parent
{
  int status;
  pid_t pid = wait(&status);
    printf("Child %d terminated within %d seconds com estado %d.\n", pid,
delay, status);
  exit(0);
}
```

Solução;
O código do filho roda por 5 segundos (sleep(5) é chamado na função sleep5), e como o delay do pai era de 7s, então o código vai finalizar sinalizando que o comando filho acabou dentro do limite.

D)
Estrutura do programa;
Função main -

```
int delay = 7; //Botando delay do pai manualmente

int main (int argc, char *argv[]){

  pid_t pid;

    signal(SIGCHLD, childhandler); //sinal que o filho passa pro pai quando
termina

  if ((pid = fork()) < 0){
    printf("Erro ao criar filho\n");
    exit(-1);
  }

  if (pid == 0) { //child
    printf("%s\n", argv[1]);
    execvp(argv[1], argv); // Passamos pro execvp a funcao sleep15
  }

  else {// parent

    printf("Delay: %d\n", delay);
      sleep(delay); // delay que o pai espera a execucao do filho, esse delay
substitui o papel do waitpid. Se passar do tempo, o pai mata o filho ao inves
de esperar ele acabar
    printf("Program %s exceeded limit of %d seconds!\n", argv[1], delay);
    kill(pid, SIGKILL);
  }
    return 0;
}
```

Função childHandler - Espera filho ser encerrado, e exibe mensagem caso tenha sido antes do fim da execução do pai

```
void childhandler(int signo) // Executed if child dies before parent
{
  int status;
  pid_t pid = wait(&status);
    printf("Child %d terminated within %d seconds com estado %d.\n", pid, delay, status);
  exit(0);
}
```

Solução;

O código do filho roda por 15 segundos (sleep(15) é chamado na função sleep15), e como o delay do pai era de 7s, então o filho vai ser interrompido pelo pai.

EX4

Objetivo;

O objetivo do programa é executar uma divisão por 0 tanto de tipo float quanto inteiro, e aguardar um sinal de erro de floating point.

Estrutura do programa;

Função main -

```
int main (int argc, char *argv[]){
  //float num1,num2;
  int num1,num2;
  printf("Digite dois numeros:");
  scanf("%d %d",&num1,&num2);

  // Chama o sinal SIGCHLD para ligar o alerta
  signal(SIGFPE, childhandler);

  /*
  printf("%f\n", num1);
  printf("%f\n", num2);
  printf("%f\n", num1 + num2);
  printf("%f\n", num1 - num2);
  printf("%f\n", num1 * num2);
  printf("%f\n", num1 / num2);
  */

  printf("%d\n", num1);
  printf("%d\n", num2);
```

```
  printf("%d\n", num1 + num2);
  printf("%d\n", num1 - num2);
  printf("%d\n", num1 * num2);
  printf("%d\n", num1 / num2);
  //printf("Program %s exceeded limit of %d seconds!\n", argv[1]);

}
```

Função childHandler - Aguarda receber um sinal do tipo de exceção de floating point.

```
void childhandler(int signo) // Executed if Floating-point exceptions
{
  //printf("Tentou calcular divisao de float com 0");
  printf("Tentou calcular divisao de intero com 0");
  exit(0);
}
```

Solução;

Código quando tenta calcular um número infinito (div por 0), mas as variáveis são inteiras, dá erro e chama o sinal (Floating-point exceptions). Quando as variáveis são float, porém, o cálculo ocorre normalmente, e o valor passa a ser "inf", que é um valor suportado pelo formato float, e nenhum erro acontece.

EX5

Objetivo;

O objetivo do enunciado é criar dois filhos de um processo, que exibirão em loop o número ao qual o processo pertence, e no processo pai alternar a execução desses dois filhos.

Estrutura do programa;

```
int delay = 5; // Botando delay do pai manualmente

int main(int argc, char *argv[]){
   pid_t pid1, pid2;
   int status;

   if ((pid1 = fork()) < 0){
      fprintf(stderr, "Erro ao criar filho\n");
      exit(-1);
   }
   else if (pid1 == 0){
      execlp("./ex5p1", "ex5p1", NULL);
      perror("execlp");
```

```
    exit(EXIT_FAILURE);
}

if ((pid2 = fork()) < 0){
    fprintf(stderr, "Erro ao criar filho\n");
    exit(-1);
}
else if (pid2 == 0){
    execlp("./ex5p2", "ex5p2", NULL);
    perror("execlp");
    exit(EXIT_FAILURE);
}

// parent
sleep(2);
printf("Parando processos ====================\n");

if (kill(pid1, SIGSTOP) == -1){
    perror("kill");
    exit(EXIT_FAILURE);
}
if (kill(pid2, SIGSTOP) == -1){
    perror("kill");
    exit(EXIT_FAILURE);
}

for (int i = 5; i >= 0; i--){
    printf("Child 1 iniciate: %d\n", pid1);
    if (kill(pid1, SIGCONT) == -1){
        perror("kill");
        exit(EXIT_FAILURE);
    }
    sleep(2);
    if (kill(pid1, SIGSTOP) == -1){
        perror("kill");
        exit(EXIT_FAILURE);
    }
    printf("Child 1 stopped: %d\n\n", pid1);

    printf("Child 2 iniciate: %d\n", pid2);
    if (kill(pid2, SIGCONT) == -1){
        perror("kill");
        exit(EXIT_FAILURE);
    }
    sleep(2);
    if (kill(pid2, SIGSTOP) == -1){
        perror("kill");
        exit(EXIT_FAILURE);
```

```
        }
        printf("Child 2 stopped: %d\n\n", pid2);
    }

    return 0;
}
```

Solução;

Ao executar o programa, podemos ver perfeitamente como que funciona a interrupção forçada dos processos, para cada um dos programas auxiliares exibindo a qual processo ele representa.