Herança de sinais com *fork()*

- Obs: Como os processos filhos recebem a imagem da memória do pai, acabam herdando o tratamento de sinais já estabelecido. Exemplo:

heranca_sinais.c

```c
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
void fin() {
    printf("SIGINT para os processos %d\n",getpid()) ;
    exit(0) ;
}
int main() {
    signal(SIGQUIT,SIG_IGN) ;
    signal(SIGINT,fin) ;
    if (fork()>0){
        printf("processo pai : %d\n",getpid()) ;
        while(1) {sleep(1);}
    }
    else {
        printf("processo filho : %d\n",getpid()) ;
        while(1) {sleep(1);}
    }
    exit(0);
}
```

Exemplo de Comunicação Com Sinais

sinais.comunicacao.c

```c
void it_fils() {
    printf("[%d] Processo filho recebe sinal SIGUSR1 e envia para si mesmo!
\n", getpid()) ;
    kill (getpid(),SIGINT) ;
}
void fils() {
    signal(SIGUSR1,it_fils) ;
    printf("[%d] Processo filho iniciado!\n", getpid()) ;
    while(1);
}
int main() {
    int pid ;
    if ((pid=fork())==0)
        fils();
    else {
        sleep(2) ;
        printf("[%d] Processo Pai executando apos o Fork.\n", getpid());
        kill (pid,SIGUSR1) ;
        sleep(1);
    }
    printf("[%d] Processo Pai encerrando.\n", getpid());
    exit(0);
}
```

Exercício 1 - Monitoramento

- Criar uma aplicação com as seguintes características
  1. Loop Principal com um contador que inicia em 0 e é incrementado até 100. Utilize a função sleep() para manter a contagem lenta.
  2. Implementar o tratamento do sinal **SIGUSR1**. Ao receber este sinal o processo deve imprimir na tela do usuário o valor atual do contador juntamente com a data e hora atual.
     2.1. Para data e hora, usar a biblioteca ***<time.h>***
     - **long** t_date;
     - **time**(&t_date);
     - **printf**("%s ", **ctime**(&t_date)) ;

Outras funções

- ***popen()***:
  - Pipe Stream de um processo para outro:
    - Cria um PIPE unidirecional;
    - Faz um Fork()
    - Chama o Shell para abrir um outro processo no lugar do filho;

```
FILE *popen(const char *command, const char *type);
```

- Uso:
  - **FILE***: Retorna um ponteiro para o ***stream*** aberto;
  - **const char *command**: Comando no formato string com o nome do binário a ser executado e seus parâmetros;
  - **const char *type**: pipe para leitura ou escrita (“r" ou “w")

Outras funções

- ***pclose()***:
  - Fecha o Pipe Stream de um processo para outro

```
int pclose(FILE *stream);
```

  - Uso:
    - **FILE* stream**: ponteiro para stream aberto;

Outras funções

- ***fileno()***:
  - Retorna o número do descritor do arquivo

```
int fileno(FILE *stream);
```

  - O numero do descritor é usado, por exemplo pelas funções de sistema ***read***, ***write***;

Exercício 2 - Comunicação com PIPE

- Criar uma aplicação com as seguintes características
  1. Abrir e ler o conteúdo de um arquivo de áudio no formato WAV;
  2. Salvar o conteúdo do arquivo em um buffer de memória;
  3. Usar a função ***popen()*** para abrir um outro processo chamado ***aplay*** e criar um PIPE de escrita entre o processo pai e o novo processo filho “aplay”.
  4. Usar a função ***fileno()*** para obter o descritor do PIPE aberto anteriormente.
  5. Usar a função ***write()*** para escrever os dados do buffer (contendo o arquivo de áudio).