Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Aula Prática 01 - Processos

Identificação da Disciplina

Código da Disciplina	Nome da Disciplina	Turma	Professor	Período
117935	Programação Concorrente	A	Jeremias Moreira Gomes	2019/2

Objetivo da Aula Prática

O objetivo desta aula prática é dar familiaridade ao aluno na manipulação básica de processos do Sistema Operacional UNIX.

Detalhes Acerca das Aulas Práticas

0.1. Restrições deste Trabalho Prático

Durante a escrita de quaisquer códigos, tentar não utilizar bibliotecas exóticas que não sejam as disponíveis no UNIX como biblioteca padrão. Se o fizer (pois ajuda na realização de testes, por exemplo), lembrar que no momento da correção o sistema padrão a ser utilizado é similar a uma instalação padrão das máquinas do LINF, e nada além do código será instalado apenas para uma correção individual. Então o uso de uma `<gtest.h>`, por exemplo, pode acabar fazendo com que o aluno perca pontos, porque seu código não pôde ser testado, em virtude da falta desta no computador de testes.

0.2. Por onde entregar os exercícios das aulas práticas?

As atividades envolvendo as aulas práticas deverão ser entregues via Aprender (https://aprender.ead.unb.br), na disciplina de Programação Concorrente. As informações para ingresso são as seguintes:

- URL: https://aprender.ead.unb.br/course/view.php?id=6775
- Chave de Acesso: `s3nhaD3ss3semestre201902`

Após o ingresso na disciplina, haverá uma atividade chamdada “Aula Prática 01 - Processos Tarefa”, para submissão dos exercícios.

0.3. O que deverá ser entregue, referente as aulas práticas?

Deverão ser entregues respostas referentes a todos os tópicos das **Seções de Atividades** ao longo deste documento. Essas atividades (dessa aula prática) estão divididas em duas categorias:

- Questionários (pergunta e resposta).
- Elaboração de Códigos.

Todos os dois tipos de atividades deverão ser entregues em um documento único contendo todas as respostas. Esse documento deverá ter identificação do aluno (nome e matrícula), identificação da disciplina,

identificação da aula prática e as respostas identificadas de maneira igual as numerações em que aparecem neste documento. Para auxiliar na elaboração desse documento, pode-se utilizar esse documento de referência.

Além disso, as questões de elaboração de código deverão ser entregues em arquivos (.c) separados, sendo um arquivo para cada código elaborado. O início desse código deverá vir com comentários fazendo referência ao autor do código, nome do arquivo e a identificação da atividade, da seguinte forma:

```
// autor: Jeremias Moreira Gomes
// arquivo: exemplo-arquivo.c
// atividade: 0.0.0

#include <stdio.h>

int main()
{
    printf("\n");
    return 0;
}
```

Assim, os códigos elaborados irão estar em arquivos separados e no relatório.

0.4. Como entregar as atividades das aulas práticas?

A submissão das atividades deverá ser feita em um arquivo único comprimido no tipo zip (`Zip archive data, at least v1.0 to extract`) contendo um diretório com o relatório e os códigos elaborados durante a atividade. Além disso, para garantir a integridade do conteúdo entregue, o nome do arquivo comprimido deverá possuir duas informações (além da extensão `.zip`):

- A matrícula do aluno.
- O *hash* md5 do arquivo `.zip`.

Para gerar o md5 do arquivo comprimido, utilize o comando `md5sum` do Linux e em seguida faça o renomeamento utilizando o *hash* coletado. Exemplo:

```
[6189] j3r3mias@tardis:aula-01-processos|master > zip -r aaa.zip atividade-01/
  adding: atividade-01/ (stored 0%)
  adding: atividade-01/relatorio.docx (stored 0%)
  adding: atividade-01/01-hello-3-fork.c (deflated 34%)
  adding: atividade-01/03-exemplos.c (deflated 47%)
  adding: atividade-01/02-arvore.c (deflated 35%)
  adding: atividade-01/02-pid_t.c (deflated 39%)
  adding: atividade-01/01-hello-fork.c (deflated 21%)
  adding: atividade-01/03-processos-e-ordens.c (deflated 55%)
[6190] j3r3mias@tardis:aula-01-processos|master > ls -lha aaa.zip
-rw-rw-r-- 1 j3r3mias j3r3mias 2,5K set  4 23:26 aaa.zip
[6191] j3r3mias@tardis:aula-01-processos|master > md5sum aaa.zip
fe36b6799eae103a464cbc4857fce404  aaa.zip
[6192] j3r3mias@tardis:aula-01-processos|master > mv aaa.zip 160068444-fe36b6799eae103a464cbc4857fce404.zip
[6193] j3r3mias@tardis:aula-01-processos|master > ls -lha 160068444-fe36b6799eae103a464cbc4857fce404.zip
-rw-rw-r-- 1 j3r3mias j3r3mias 2,5K set  4 23:26 160068444-fe36b6799eae103a464cbc4857fce404.zip
[6194] j3r3mias@tardis:aula-01-processos|master > md5sum 160068444-fe36b6799eae103a464cbc4857fce404.zip
fe36b6799eae103a464cbc4857fce404  160068444-fe36b6799eae103a464cbc4857fce404.zip
[6195] j3r3mias@tardis:aula-01-processos|master >
```

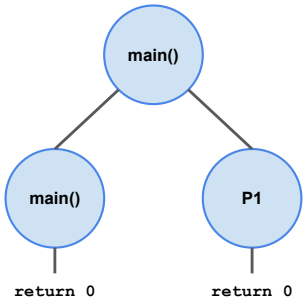
1. *Hello Fork*

A ideia do primeiro exercício desta lista, é dar uma visualização do funcionamento da função `fork`. O arquivo de cabeçalho que provê acesso as principais chamadas de sistemas é o `<unistd.h>`. Além desta, também podem ou vão ser utilizados as bibliotecas `<stdlib.h>`, `<sys/types.h>` e `<sys/wait.h>`. Então para escrevermos o programa `hello-fork.c`, podemos fazer da seguinte forma:

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

int main()
{
    fork();
    printf("Hello, fork!\n");
    return 0;
}
```

Neste código, a função `fork` cria um clone exato do processo que a chamou. Dessa forma, podemos visualizar o fluxo de execução desse processo da seguinte forma:



1.1. Atividade

1.1.1. Contagem de Saídas

Dado o trecho de código abaixo, quantas vezes o texto “Hello, fork!” irá aparecer na tela?

```
fork();
fork();
fork();
printf("Hello, fork!\n");
```

1.1.2. Hierarquia de Processos

Desenhe o fluxo de execução, a partir do trecho de código apresentado no item anterior, em forma de hierarquia de processos.

1.1.3. Similar ao trecho de código anterior, para um número n de *forks*, quantos processos são criados ao todo?

2. Identificação de Processos

Quando um processo é criado utilizando a função `fork`, é retornado o valor do tipo `pid_t`, cujo valor representa o *Process Indentifier* (PID) do processo criado. Lembrando que na utilização do `fork`, duas linhas de execução são criadas. Então, no processo pai, o `pid_t` será o PID do filho, e no processo filho (que foi criado) o PID será 0 (zero)[1]. Dessa forma, é possível diferenciar a execução a partir do `fork` utilizando condicionais.

```
pid_t pid;
pid = fork();
if (pid != 0) {
    printf("%ld, I am your father.\n", pid);
} else {
    printf("NOOOOOOOOOOOOOOOOOOOO\n");
}
```

Lembre-se sempre que não é possível assumir qual dos processos irá imprimir primeiro, então para este trecho apresentado, duas ordens para a saída são possíveis. Teste.

2.1. Atividade

Nomeando a `main` como $P0$, desenhe o fluxo de execução dos processos, para o seguinte trecho:

```
fork();
fork() && (fork() || fork());
fork();
putchar('.');
```

Obs: Não é necessário colocar as chamadas `return` no desenho.

3. Outras Chamadas de Sistema

Além da função `fork`, algumas outras funções são bem comuns ao se manipular o uso de processos. A lista abaixo apresenta algumas dessas funções, com uma descrição sumária de seu funcionamento.

- `getpid()` - Retorna o número do processo em execução.
- `getppid()` - Retorna o número do pai do processo em execução.
- `exit()` - Termina o processo que o invocou (padrão da biblioteca C).

[1] Se o pid retornado para o processo pai for < 0, houve um problema e o processo não foi criado

- `vfork()` - Cria um processo filho e bloqueia o processo pai até que este termine sua execução (`exit`).

- `wait()` - Força o processo pai esperar até que um filho pare ou termine a sua execução.

- `waitpid()` - Espera por um filho específico terminar sua execução[2].

- `exec` - Família de funções que criam um novo processo substituindo completamente a imagem do processo que o criou.

  - `execl(const char *path, const char *arg, ...);`

  - `execlp(const char *file, const char *arg, ...);`

  - ```
    execle(const char *path, const char *arg , ...,
            char* const envp[]);
    ```

  - `execv(const char *path, char *const argv[]);`

  - `execvp(const char *file, char *const argv[]);`

[2]Para verificar a sintaxe de utilização, utilizar o comando (via shell) *man waitpid*
    ```

Exemplo de utilização de algumas dessas chamadas.

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

int main()
{
  pid_t pid = fork();
  printf("Meu pid é: %d\n", getpid());
  if (pid) {
    wait(NULL);
    printf("Processo filho terminou.\n");
  } else {
    printf("Filho saindo...\n");
    exit(0);
  }

  printf("Irei imprimir somente uma vez, pois o filho já terminou.\n");

  execl("/bin/ls","ls",".", NULL) ;

  printf("Meu processo está morto! Tens o que é necessário para imprimir?");

  return 0;
}
```

3.1. Atividades

3.1.1. Escreva um programa para receber um inteiro n ($0 < n < 10$) e criar especificamente essa quantidade de processos. Cada um dos processos filhos deverá imprimir o seu PID e o PID do seu pai, que deve ser o mesmo para todos os processos filho. A frase impressa deverá ser a seguinte: "Sou o processo %d e o PID do meu criador é %d.". Não esquecer de acrescentar uma quebra de linha ao final da frase.

3.1.2. Escreva um programa cujos eventos de cada processo deverão seguir a ordem em que aparecem abaixo:

- O processo pai deverá criar um processo ($p1$).
- O pai, e somente ele, irá criar um segundo processo ($p2$).
- O pai deverá imprimir o texto "Sou pai de dois.".
- O pai irá esperar o filho $p1$ terminar normalmente a sua execução.
- O pai deverá imprimir "Acho que meus filhos terminaram suas execuções.".

- O processo filho $p1$ deverá dormir por 2 segundos (função `sleep`) e imprimir "Sou o filho mais velho e dormi um pouco.".
- O processo filho $p1$ deverá imprimir a frase "Sou o filho %d, e estou terminando agora.", onde %d é o PID do processo.
- O processo filho $p2$ deve imprimir a frase "Sou o filho mais novo.".
- O processo filho $p2$ deverá imprimir a frase "Sou o filho %d, e estou terminando agora.", onde %d é o PID do processo (mesma frase que o processo $p1$ irá imprimir).

3.1.3. Ao executar um programa e invocar como primeira função a chamada $getppid$, qual o nome do processo/programa cujo número representa esse PID retornado?