Disciplina: Sistemas Operacionais I

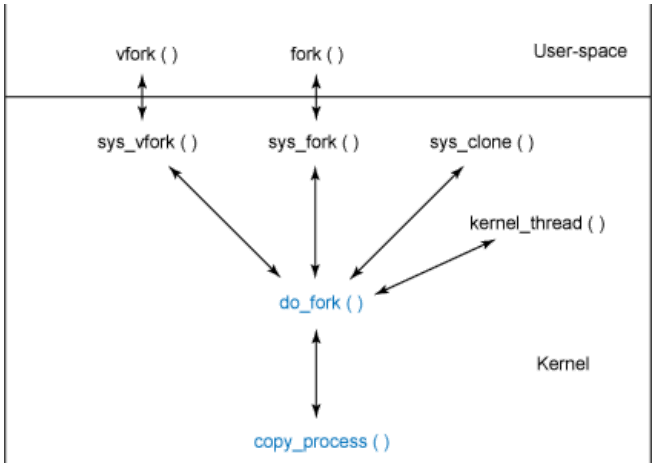
Exercício 1

1. Elabore um programa em Linux que crie um processo e execute um segundo programa.

    a. Explique o código fonte utilizado

```
#include <unistd.h> //Aqui fica a função exec()
#include <stdio.h>
#include <sys/types.h> //Aqui fica a função fork()
#include <sys/wait.h>

using namespace std;

int main( ){
   //pid_t é um tipo inteiro, usado para conseguir usar o getpid() e o getppid()
   pid_t child_pid;
N
   // Cria um processo filho que é a cópia do pai, inclusive está cópia executa com o
ponteiro no mesmo lugar em que o fork() foi executado
   child_pid = fork ();

   //Essa estrutura de If's consegue identificar se o comando falhou ou se é o filho pelo
Return do fork(), pois no processo filho ele retorna 0, ou negativo em caso de erro.
   if (child_pid < 0) {
      printf("O programa falhou em executar o comando fork()");
      return 1;

   } else if (child_pid == 0) {
      //Fiz um sleep para que seja possível enxergar que o processo pai está esperando o
processo filho
      sleep(5);

      //Aqui eu irei abrir o Google Chrome, usando o $PATH (O caminho do Google Chrome
está guardado lá), assim eu só tenho que passar o nome do programa. Passei o site como
segundo argumento, para abrir no site que eu quero.
      char *args[]={"google-chrome","www.facebook.com",NULL};

      //Da família Exec (existem muitas variações), o 'v' significa que eu vou passar um
vector, que é o *args[] e o 'p' é de path, ou seja terá um caminho para o novo programa.
      execvp(args[0],args);

   } else {
      wait(NULL);
   }
   return 0;
}
```

b. Descreva os resultados obtidos



Todos os forks são remanescentes de uma função chamada do_fork(), no Kernel do Linux.

*"A função **do_fork** começa com uma chamada para **alloc_pidmap**, que aloca um novo PID. Em seguida, **do_fork** faz a verificação para ver se o depurador está rastreando o processo pai. Se estiver, o sinalizador **CLONE_PTRACE** será configurado em **clone_flags** na preparação para a bifurcação. A função **do_fork** continua então com uma chamada para **copy_process**, transmitindo os sinalizadores, a pilha, os registros, o processo pai e o PID recentemente alocado."*

"A process created using the UNIX fork() function is expensive in setup time and memory space. In fact it is sometimes called a heavyweight process. Properties of a heavyweight process are:

1. *Heavyweight processes run independently and do not share resources*
2. *They consist of code, stack, and data "*

O fato do fork() ser uma cópia do processo, aparenta ser muito custoso, e essa impressão é verdadeira. Mas há benefícios em fazer cópia, principalmente a divisão de recursos, escopo e a possibilidade do processo pai alterar o processo filho antes de executá-lo. O momento certo de usar Threads e processos é o segredo. Bem como as versões mais recentes do Kernel, permite criar processos com uma divisão diferente da memória.

Referências:

-https://www.ibm.com/developerworks/br/library/l-linux-process-management/index.html

-https://www.infowester.com/linprocessos.php

2. Elabore um programa em Windows que crie um processo e execute um segundo programa.
    a. Explique o código fonte utilizado

```
#include <Windows.h>

//LPCSTR significa Long Pointer to Const String, ou seja, constant null-terminated
string of CHAR.
void startup(LPCSTR nome_aplicacao)
{
    // Variáveis para informação adicional
    STARTUPINFOA si;
    PROCESS_INFORMATION pi;

    // Aloco o tanto de memória que as estruturas vão precisar
    ZeroMemory(&si, sizeof(si));
    si.cb = sizeof(si);
    ZeroMemory(&pi, sizeof(pi));

    // Crio o Processo
    CreateProcessA
    (
        nome_aplicacao,   // Caminho que eu passei como parâmetro para a função
        argv[1],                // Argumento da linha de comando
        NULL,                   // O Gerenciamento do processo não é herdável
        NULL,                   // O Gerenciamento da Thread não é herdável
        FALSE,                  // Gerenciamento de heranças FALSE
        CREATE_NEW_CONSOLE,     // Abra os arquivos em um console diferente
        NULL,           // Uso do ambiente do pai
        NULL,           // Uso do diretório primário do pai
        &si,            // Ponteiro para a estrutura STARTUPINFO
        &pi           // Ponteiro para a estrutura PROCESS_INFORMATION
    );
        // Espera o processo filho terminar
        WaitForSingleObject( pi.hProcess, INFINITE );

        // Fecha o processo e seus gerenciadores.
        CloseHandle(pi.hProcess);
        CloseHandle(pi.hThread);
}
```

b. Descreva os resultados obtidos

O Windows te dá muitas opções para criação dos processos, e isso muitas vezes te obriga a ser um “Geek de configuração” para conseguir fazer um bom uso para os processos. Nesse caso, eu abro um programa enviado pelo usuário e isso é uma ótima alternativa para o System(), como eu explico logo abaixo.

Aqui estão alguns motivos de porque você deveria usar CreateProcess e evitar o uso do System():

- Ele é um recurso extremamente pesado
- Ele enfraquece a segurança - Você não sabe se o que está sendo executado é um comando válido ou se ele age da mesma forma em todos os sistemas, você pode iniciar programas que você não possuía a intenção de executar. O perigo está quando você executa um programa diretamente, pois ele possui o mesmo privilégio que o programa que o executou - Significa que, por exemplo, se você roda o programa como administrador, então o programa malicioso que você executou sem querer também terá acessos de administrador.
- Os programas de Antivírus normalmente colocam flag de Vírus para programas que usam esse tipo de função.

Referências:
-https://docs.microsoft.com/en-us/windows/win32/procthread/creating-processes
-https://docs.microsoft.com/en-us/windows/win32/procthread/about-processes-and-threads

3. Elabore um programa em Java que crie um processo e execute um segundo programa.
    a. Explique o código fonte utilizado

Classe do Processo:

```
//Runtime pois vai rodar em tempo de execução
Runtime runtime = Runtime.getRuntime();
//Então eu crio um processo e associo à um processo executado em tempo de
execução, nesse caso o ENV (Environment)
Process process = runtime.exec("env");

//Aqui eu uso o Try para poder cuidar dos erros, e nesse caso eu vou pegar o
InputStream() do processo
try (InputStream in = process.getInputStream();) {
    //Associo um Byte[] para guardar as informações e um len para o tamanho
    byte[] bytes = new byte[2048];
    int len;
    //Vou imprimir algumas informações do ambiente
    while ((len = in.read(bytes)) != -1) {
    System.out.write(bytes, 0, len);
    }
    //Aqui eu vou abrir o Google Chrome na página do Facebook.
    Runtime.getRuntime().exec("google-chrome ['www.facebook.com']");
}
System.out.println("Process exited with: " + process.waitFor());
```

b. Descreva os resultados obtidos

Como o Java é uma linguagem e tem sua própria máquina virtual, o JVM, isso permite que ele construa seus processos bem semelhantes à como você constrói programas na Linguagem. E nesse caso, Threads e Processos estão bem interligados.

4. Elabore um programa em Python que crie um processo e execute um segundo programa.
   a. Explique o código fonte utilizado

```
import os # Importo a biblioteca

def child():
   print('\nUm novo processo filho ',  os.getpid())
   subprocess.Popen(['C:\Program Files\Mozilla Firefox\\firefox.exe', '-new-tab'])
# Abro o Firefox no processo filho
   os._exit(0) # Fecho o processo

def parent():

   # coloco um while(true) para poder testar várias vezes
   while True:
      newpid = os.fork() # Fork semelhante ao do linux

      # Note que a estrutura é muito parecida com a do Linux
      if newpid == 0:
         child()
      else:
         pids = (os.getpid(), newpid)
         print("parent: %d, child: %d\n" % pids)
      # Aqui eu continuo o loop
      reply = input("q para sair / c para um novo Fork()")
      if reply == 'c':
          continue
      else:
          break

parent()
```

b. Descreva os resultados obtidos

Em Python, a linguagem constrói processos de forma semelhante ao Linux (Pois a linguagem tem sua base em C). Em Python a simplicidade é destacada.