Universidade do Algarve

Faculdade de Ciências e Tecnologia

Licenciatura em Engenharia Informático

Sistemas Operativos

Relatório 2º Guião Prático

Discente: Hugo Paixão, nº 64514

Docentes: Amine Berqia e José Bastos

22 de Abril de 2021

Índice

1. Nota Prévia

Como forma de facilitar a leitura do código, este vai estar em forma de printscreen, tal como a sua execução.

2. Introdução

2.1 Posix

POSIX (Portable Operating System Interface) é uma família de padrões especificados pela IEEE Computer Society para manter a compatibilidade entre sistemas operativos. POSIX define a interface de programação de aplicativos (API), junto com a linha de comandos shell e interfaces de utilitários, para compatibilidade de software do Unix e outros sistemas.

2.2 Chamadas de Sistema

System Call ou syscall, é a maneira pela qual um programa solicita um serviço do kernel do sistema operativo no qual é executado. Isto pode incluir serviços relacionados a hardware, criação e execução de novos processos e comunicação com serviços kernel integrais, como agendamento de processos. As System calls fornecem uma interface essencial entre o processo e o sistema operacional. A System Call tem como funções a gerência de processos e threads, a gerência de memoria, a gerência do sistema de arquivos e a gerência de dispositivos.

2.3 Processos

Um processo é basicamente um programa em execução. Associado a cada processo está o espaço de endereço, uma lista de posições de memoria a partir de um mínimo, até um máximo que o processo pode ler e escrever. O espaço de endereços contém o programa executável, os dados do programa e a sua pilha. Também associados a cada processo está um conjunto de registadores, incluindo o contador de programa, o ponteiro e outros registadores de hardware e todas as outras informações necessárias para a execução do programa.

2.4 Fork()

O fork() é usado para criar um novo processo em um sistema do tipo Unix. Quando criamos um processo por meio do fork(), dizemos que este processo é filho, e o processo pai é aquele que chamou o fork(). Ao usar o fork(), será criado um processo filho, que será idêntico ao pai inclusive tendo as mesmas variáveis, registos, descritores de arquivos, etc. Ou seja, o processo filho é uma copia do pai, "exatamente igual". Na verdade, não será exatamente igual já que as informações de controle serão diferentes, como o caso do pid e ppid.

2.5 Pipe()

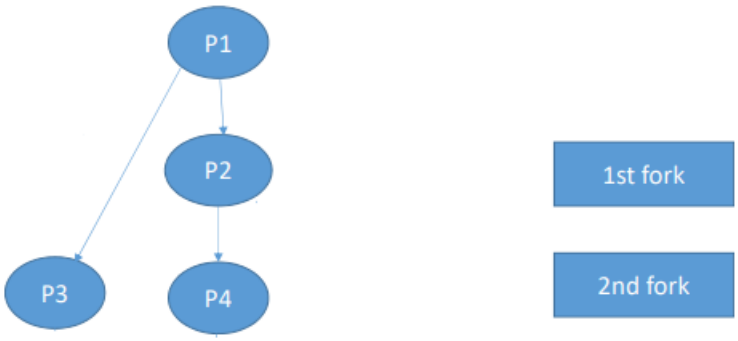
Pipe estabelece uma cominuição entre dois processos, um de um de cada lado do pipe. Um processo escreve (write()) o dado de um lado e outro lê (read()) do outro. Uma comunicação direta entre eles. Para a criação do pipe são necessários os file descriptors, que nada mais são que dois inteiros que vão definir a entrada-padrão e a saída-padrão do pipe. Precisaremos apenas de uma coisa, um vetor de inteiros de duas posições. O primeiro elemento (0) define o vetor de leitura (saída) de dados e o segundo elemento (1), define a escrita de dados no pipe (entrada);

Ambas as funções write e read, têm três parâmetros, o file do descriptor, para especificar o pipe. O argumento que passamos para essas chamadas de sistema é um endereço de memoria, ou seja, passamos um ponteiro. O terceiro é o número de bytes, para a partir do endereço de memoria quantas informações vamos ler/escrever pelo pipe.

Concurso P2:

Exercício 1:

1) São iniciados 4 processos. Caso estejam N instruções fork() seguidas o número total de processos formados será de 2^N.



2) O fork vai retornar 0, e como pid = fork(), o que está dentro do if será um 0 que é igual a falso, logo o programa como o if e falso vai saltar para o else. E apos o fork() retornar o id do pai este irá esperar que os processos filhos acabem de executar para correr.

3) O output deste programa são todos os processos ativos no programa, que são 4 processos. O primeiro fork() antes do &&, corre quando apenas o processo pai. Dentro dos parentes, o || corre o processo pai e filho do primeiro e o processo pai do segundo contando assim um total de 4 processos.

```
  PID TTY          TIME CMD
11148 pts/0    00:00:00 bash
11948 pts/0    00:00:00 a.out
11949 pts/0    00:00:00 a.out
11950 pts/0    00:00:00 a.out
11951 pts/0    00:00:00 sh
11952 pts/0    00:00:00 a.out
11953 pts/0    00:00:00 sh
11954 pts/0    00:00:00 ps
11955 pts/0    00:00:00 ps
  PID TTY          TIME CMD
11148 pts/0    00:00:00 bash
11948 pts/0    00:00:00 a.out
11950 pts/0    00:00:00 a.out
11952 pts/0    00:00:00 a.out
11953 pts/0    00:00:00 sh
11955 pts/0    00:00:00 ps
  PID TTY          TIME CMD
11148 pts/0    00:00:00 bash
11948 pts/0    00:00:00 a.out
11950 pts/0    00:00:00 a.out
11956 pts/0    00:00:00 sh
11957 pts/0    00:00:00 ps
  PID TTY          TIME CMD
11148 pts/0    00:00:00 bash
11958 pts/0    00:00:00 sh
11959 pts/0    00:00:00 ps
11948 pts/0    00:00:00 a.out
11950 pts/0    00:00:00 a.out
11956 pts/0    00:00:00 sh
11957 pts/0    00:00:00 ps
11958 pts/0    00:00:00 sh
11959 pts/0    00:00:00 ps
```

Figura 1 - Resultados da execução do 3

4) O output deste programa são todos os processos ativos no programa, que são 3 processos. O primeiro fork() antes do &&, corre apenas o processo pai, e o segundo fork() corre o processo pai e filho totalizando assim 3 processos.

Figura 2 - Resultado da execução do 4

```
  PID TTY          TIME CMD
11148 pts/0    00:00:00 bash
  PID TTY          TIME CMD
12775 pts/0    00:00:00 a.out
11148 pts/0    00:00:00 bash
12776 pts/0    00:00:00 a.out
12777 pts/0    00:00:00 a.out
12778 pts/0    00:00:00 sh
12779 pts/0    00:00:00 sh
12780 pts/0    00:00:00 ps
12781 pts/0    00:00:00 ps
12775 pts/0    00:00:00 a.out
12776 pts/0    00:00:00 a.out
12777 pts/0    00:00:00 a.out
12778 pts/0    00:00:00 sh
12779 pts/0    00:00:00 sh
12780 pts/0    00:00:00 ps
12781 pts/0    00:00:00 ps
  PID TTY          TIME CMD
11148 pts/0    00:00:00 bash
12775 pts/0    00:00:00 a.out
12777 pts/0    00:00:00 a.out <defunct>
12782 pts/0    00:00:00 sh
12783 pts/0    00:00:00 ps
```

Exercício 2:

1.

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

int main() {
  pid_t pid = fork();
  if(pid) {
    sleep(1);
    for(int i = 0; i < 5; i++)
      printf("Eu sou o pai, minha identificação ém %d.\n", getpid());
  } else {
    for(int j = 0; j < 3; j++)
      printf("Eu sou o filho, meu pai é %d.\n", getppid());
  }
  exit(0);
}
```

```
hugopaixao@hugopaixao-VirtualBox:~/Desktop/SO/Pratica/LabsEntrega/Lab2$ gcc -Wall test.c
hugopaixao@hugopaixao-VirtualBox:~/Desktop/SO/Pratica/LabsEntrega/Lab2$ ./a.out
Eu sou o filho, meu pai é 13899.
Eu sou o filho, meu pai é 13899.
Eu sou o filho, meu pai é 13899.
Eu sou o pai, minha identificação ém 13899.
Eu sou o pai, minha identificação ém 13899.
Eu sou o pai, minha identificação ém 13899.
Eu sou o pai, minha identificação ém 13899.
Eu sou o pai, minha identificação ém 13899.
hugopaixao@hugopaixao-VirtualBox:~/Desktop/SO/Pratica/LabsEntrega/Lab2$
```

2.

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

int main() {
  printf("Eu sou o pai grande %d.\n", getpid());
  pid_t pid = fork();
  if(pid > 0) {
    for(int i = 0; i < 2; i++) {
      if(fork() == 0) {
        printf("Eu sou o filho %d, meu pai é %d\n", getpid(),getppid());
        exit(0);
      }
    }
  } else {
    for(int i = 0; i < 2; i++) {
      if(fork() == 0) {
        printf("Eu sou o filho %d, meu pai é %d\n", getpid(),getppid());
        exit(0);
      }
    }
    printf("Eu sou o filho %d, meu pai é %d\n", getpid(),getppid());
  }
  while(wait(NULL) != -1);
  return 0;
}
```

```
hugopaixao@hugopaixao-VirtualBox:~/Desktop/SO/Pratica/LabsEntrega/Lab2$ gcc -Wal
l p2.c
hugopaixao@hugopaixao-VirtualBox:~/Desktop/SO/Pratica/LabsEntrega/Lab2$ ./a.out
Eu sou o pai grande 12300.
Eu sou o filho 12303, meu pai é 12300
Eu sou o filho 12302, meu pai é 12300
Eu sou o filho 12304, meu pai é 12301
Eu sou o filho 12301, meu pai é 12300
Eu sou o filho 12305, meu pai é 12301
hugopaixao@hugopaixao-VirtualBox:~/Desktop/SO/Pratica/LabsEntrega/Lab2$
```

Exercício 3:

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
#include<unistd.h>
#include<errno.h>
#include<sys/types.h>
#include<sys/wait.h>

char *changeStr(char *str, char *tmp) {
  for(int i = 0; str[i] != '\0'; i++) {
    if(str[i] >= 'A' && str[i] <= 'Z')
      tmp[i] = (char) tolower(str[i]);
    else if(str[i] >= 'a' && str[i] <= 'z')
      tmp[i] = (char) toupper(str[i]);
    else
      tmp[i] = str[i];
  }
  return tmp;
}
```

```c
int main() {
  int p[2];
  int p2[2];
  if(pipe(p) < 0)
    exit(1);
  if(pipe(p2) < 0)
    exit(1);
  pid_t pid = fork();
  while(1) {
    if(pid == 0) {
      char str[1000];
      scanf("%s", str);
      close(p[0]);
      write(p[1], str, sizeof(str));
      close(p2[1]);
      read(p2[0], str, sizeof(str));
      for(int i = 0; str[i] != '\0'; i++)
        printf("%c", str[i]);
      printf("\n");

      memset(str, 0, sizeof str);
    } else {
      char str[1000];
      char tmp[1000];
      close(p[1]);
      read(p[0], str, sizeof(str));
      changeStr(str, tmp);
      close(p2[0]);
      write(p2[1], tmp, sizeof(tmp));
      memset(tmp, 0, sizeof tmp);
      memset(str, 0, sizeof str);
    }
  }
  return 0;
}
```

```
hugopaixao@hugopaixao-VirtualBox:~/Desktop/SO/Pratica/LabsEntrega/Lab2$ gcc -Wall p3.c
hugopaixao@hugopaixao-VirtualBox:~/Desktop/SO/Pratica/LabsEntrega/Lab2$ ./a.out
wqert
WQERT
QWEWRT
qwewrt
pSpSpS1231
PsPsPs1231
.,SDS232sada
.,sds232SADA
^C
hugopaixao@hugopaixao-VirtualBox:~/Desktop/SO/Pratica/LabsEntrega/Lab2$
```

Conclusão:

Com a elaboração deste relatório foram introduzidos alguns conceitos fundamentais principalmente acerca de processos e a sua manipulação. Neste trabalho prático também adquiri alguma experiência no desenvolvimento de programas que utilizem mecanismos de sincronização e comunicação entre processos, com base na programação em C. A explicação dos resultados obtidos foi bastante importante para compreender todos estes conceitos à volta dos processos. Porque ao estar a comentar e a ver ao mesmo tempo o código compreendia melhor o objetivo dos programas, e o que cada linha fazia na execução do programa. No entanto tive algumas dificuldades, no que toca a algumas funções que eram apresentadas nos exercícios iniciais, como é o caso do fork(), getpid(), getppid(), etc. Mas, contudo, pesquisei sobre estas e obtive a informação necessária para a realização dos exercícios e para a conclusão deste trabalho prático. Em suma, com a elaboração deste relatório prático, permitiu-me introduzir alguns conceitos, e adquirir alguma experiência na manipulação e criação de processos num sistema operativo.

Webgrafia:

https://www.includehelp.com/c/process-identification-pid_t-data-type.aspx
https://www.quora.com/How-could-I-calculate-the-number-of-processes-generated-using-N-fork-statements https://www.youtube.com/watch?v=PwxTbksJ2fo

https://statics-submarino.b2w.io/sherlock/books/firstChapter/2012965934.pdf

https://en.wikipedia.org/wiki/System_call