Sistemas Operacionais – Prof. Rafael R. Obelheiro

Exercícios — IPC no Linux

1. O código abaixo* possui uma condição de disputa. Elimine essa condição de disputa do código usando mutex, de modo que o programa sempre imprima `n=0`.

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

#define MAX 2000

int n;

void f1(void *argp) {
    int i, temp;
    for (i = 0; i < MAX; i++) {
        temp = n;
        temp++;
        n = temp;
    }
}

void f2(void *argp) {
    int i, temp;
    for (i = 0; i < MAX; i++) {
        temp = n;
        temp--;
        n = temp;
    }
}

int main(void) {
    pthread_t t1, t2;
    int rc;
    n = 0;
    rc = pthread_create(&t1, NULL, (void *)f1, NULL);
    rc = pthread_create(&t2, NULL, (void *)f2, NULL);
    rc = pthread_join(t1, NULL);
    rc = pthread_join(t2, NULL);
    printf("n=%d\n", n);
    return 0;
}
```

2. Considere um programa concorrente com três *threads*, `X`, `Y` e `Z`, mostradas abaixo.

```
int n = 1;     /* variável compartilhada entre X, Y e Z */

void X() {
   n = n * 16;
}
```

```
void Y() {
   n = n / 7;
}
```

```
void Z() {
   n = n + 40;
}
```

Implemente esse programa usando Pthreads, e use variáveis de condição para garantir que o resultado final de `n` seja sempre 8.

*Exercício 6 da lista de Pthreads, disponível como `pth-6.c` no arquivo de exemplos de IPC no Linux.

3. Generalize o código do produtor-consumidor com Pthreads visto em aula (solução do Tanenbaum) para usar um buffer circular com N posições.

4. Resolva o exercício 1 com processos no lugar de *threads*, usando `fork()`, memória compartilhada e semáforos (a variável `n` deve ser compartilhada entre os processos).

5. O que acontece caso um processo tente usar uma região de memória compartilhada maior do que a alocada? Por exemplo, caso seja alocado espaço para um vetor de 1000 inteiros, o que acontece se um processo tentar usar 2000 inteiros?

6. Resolva o exercício 2 com processos no lugar de *threads*, usando `fork()`, memória compartilhada e semáforos.

7. Generalize o código do produtor-consumidor com memória compartilhada e semáforos visto em aula para usar um buffer circular com N posições.

8. Modifique a sua solução do exercício 3 para usar semáforos POSIX no lugar de mutex e variáveis de condição.