Universidade Federal de Uberlândia (UFU)

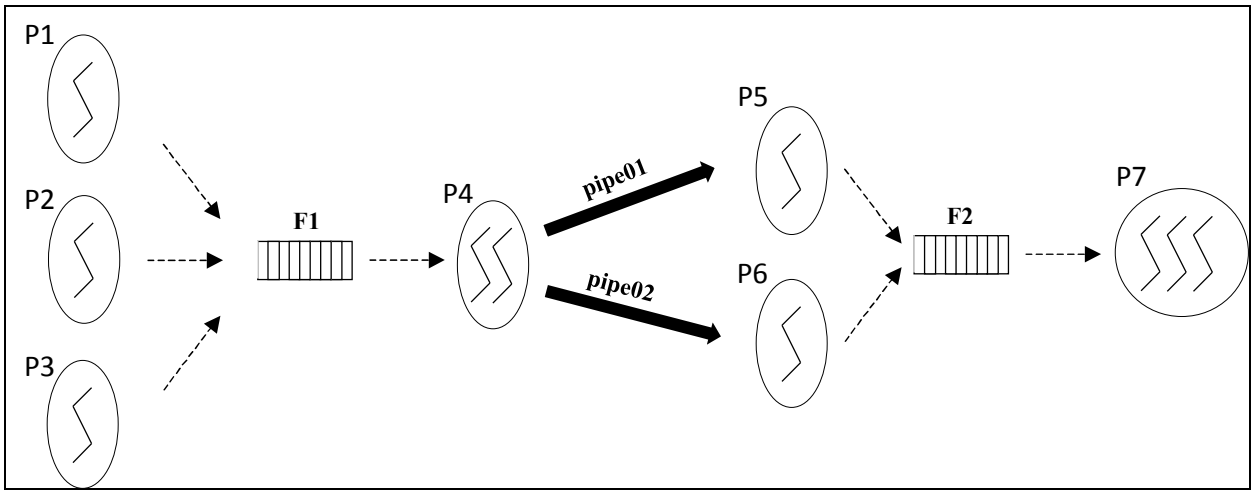
Faculdade de Computação (FACOM)

Disciplina: Sistemas Operacionais

Prof. Rivalino Matias Jr.

Bônus III (Unidade III)

1) Implementar um programa que realize o processamento ilustrado na figura abaixo:



Os processos P1, P2 e P3 são *singlethreads*. Cada um desses processos se comunica com o processo P4 por meio de uma área de memória compartilhada (*shared memory*). Essa área compartilhada deve ser usada como uma fila (F1) do tipo FIFO (*first-in-first-out*), a qual armazena valores do tipo inteiro. A capacidade da fila é de 10 valores.

Os processos P1, P2 e P3 são produtores e o processo P4 é consumidor nessa comunicação. P4 sempre aguarda um sinal (SIGUSR1) para consumir dados da F1. Esse sinal é enviado para P4 quando F1 estiver cheia, ou seja, com 10 valores. O sinal deve ser enviado para P4 pelo processo que inseriu o décimo valor em F1. Note que neste caso F1 somente receberá valores após P4 retirar o último (décimo) valor de F1, deixando-a vazia para receber novos valores.

Os valores inteiros inseridos em F1 devem ser gerados de forma aleatória na faixa de 1 até 1000.

O acesso a F1 deve ocorrer de forma exclusiva utilizando o mecanismo de semáforo entre os processos envolvidos.

O processo P4 possui duas *threads* (t1 e t2), onde ambas as *threads* retiram valores de F1 e enviam para P5 (*t1*) e P6 (*t2*), respectivamente, utilizando o mecanismo de *pipe*. Os processos P5 e P6 ao receberem valores de P4 os enviam para P7 utilizando da fila F2, também implementada como *shared memory*. Neste caso, o controle ao acesso de F2 deve ser implementado usando um mecanismo de exclusão mútua baseado em espera ocupada (*busy wait*).

Diferente de F1, a fila F2 deve ser consumida na medida em que entram novos valores. Se a fila estiver cheia (máximo 10 valores), então os processos P5 e P6 devem aguardar a retirada de ao menos um elemento da fila, por P7, para inserirem novos valores.

Todas as três *threads* de P7 retiram valores de F2 e os imprimem na tela.

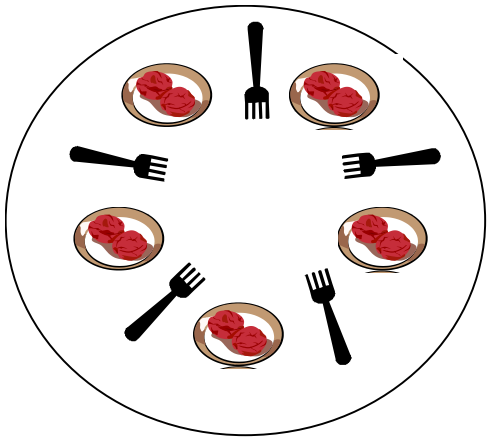
Após P7 imprimir 10000 valores, o programa deve imprimir o seguinte relatório:

a) Tempo total de execução do programa.
b) Quantidade de valores processados por P5 e P6.

c) Dentre os valores impressos, informar o valor que mais se repetiu (moda), o valor mínimo e o valor máximo.
obs:para gerar números aleatórios use

```
#include <stdlib.h>
int rand(void);
int rand_r(unsigned int *seedp);
void srand(unsigned int seed);
```

2) Fazer um programa, usando *threads*, para ilustrar o problema clássico do **Jantar dos Filósofos** (*ver livro texto da disciplina*). O desenho abaixo ilustra este cenário.



Em frente de cada prato existe um filósofo sentado, ou seja, existem 5 filósofos sentados na mesa. Para comer o seu espaguete, cada filósofo precisa de dois garfos. O comportamento do filósofo se resume a **comer** e **pensar**. Desta forma, aleatoriamente, o filósofo deve comer e pensar, indefinidamente. A ação de **comer** significa pegar dois (2) garfos, primeiro o do lado esquerdo e depois do lado direito, um em cada mão, e aguardar um determinado intervalo de tempo aleatório (rand(3)). Posteriormente, o filósofo deve deixar os garfos sobre a mesa, novamente primeiro o do lado esquerdo e depois do lado direito, a fim de pensar um pouco. A ação de **pensar** também deve ser implementada como uma inatividade por um determinado período de tempo aleatório.

Nesta implementação, propositalmente, você não deve usar qualquer mecanismo de exclusão mútua com o objetivo de experimentar o problema de inconsistência no uso dos recursos compartilhados.

3) Alterar o programa do exercício #2 para incluir o uso de exclusão mútua entre *threads* (filósofos) vizinhas que compartilham o mesmo garfo. Utilizar o mecanismo de semáforos.

4) Como pode ser observado no resultado do exercício #3, a implementação com semáforo resolverá o problema de compartilhamento de recursos, mas causará o problema de *deadlock*. Neste exercício você deverá fazer mudanças no seu código a fim de eliminar as condições de *deadlock*.