Exercícios de Revisão – IPC

1) Desenvolva um programa concorrente utilizando Filas de Mensagens que leia um vetor de 1000 posições e conte quantos números primos existem nesse vetor. A contagem dos números primos desse vetor deve ser dividida entre 2 processos filhos. O processo pai deve somente buscar os valores parciais obtidos pelos filhos e calcular e escrever na tela o valor total.

Para o desenvolvimento do programa utilize uma linguagem de programação real e as seguintes chamadas de sistema:

- *int fork()*: cria uma cópia do processo original. A chamada de sistema fork retorna o <*pid*> do filho para o processo pai e *0 (zero)* para o processo filho.
- *int createQueue(<endereco>):* cria uma fila de mensagens no endereco <*endereco*> e retorna o identificador da mesma.
- *send(fila, valor, tipo):* escreve uma mensagem contida em <*valor*> com tipo <*tipo*> na fila de mensagens <*fila*>.
- *recv (fila, valor , tipo):* remove uma mensagem do tipo <*tipo*> da fila de mensagens fila <*fila*> e armazena no endereço <*valor*> . Se tipo for *0 (zero)* remove o primeiro elemento da lista.
- *wait():* bloqueia o processo pai até que um processo filho termine.
- *freeQueue(<fila>):* destrói a fila de mensagens <*fila*>.

2) Pipes são empregados para estabelecer comunicação entre processos pai e filho. Um pipe pode ser definido como um canal unidirecional de comunicação, isto é, informação flui numa única direção. Faça um programa concorrente utilizando pipes que leia uma string e determine a freqüência de vogais no texto. Exemplo para o texto "Maguila derruba Tyson" a freqüência de vogais é 8/21. A contagem das vogais deve ser dividida igualmente entre 2 processos (pai e filho) sendo que o resultado final deve ser escrito pelo processo pai.

Para o desenvolvimento do programa utilize uma linguagem de programação real e as seguintes chamadas de sistema:

- *int fork()*: cria uma cópia do processo original. A chamada de sistema fork retorna o *pid* do filho para o processo pai e *0 (zero)* para o processo filho.
- pipe(<int fd[2]>): cria um pipe. Dois descritores são retornados *fd[0]* para leitura e *fd[1]* para escrita. Esta chamada deve ser realizada antes de ocorrer um fork.
- *close(<int fd>):* fecha um descritor. Uma vez que os pipes permitem comunicações em uma única direção os descritores não utilizados devem ser fechados. Por exemplo, se a comunicação for no sentido pai -> filho o processo pai deve fechar o descritor de leitura *(close (fd[0]))* e o processo filho deve fechar o descritor de escrita *(close(fd[1]))*;
- *read(<int fd>, <pos>):* lê o conteudo armazenado no descritor <*fd*> e armazena na variavel <*pos*>. Importante: a leitura de *pipes e bloqueante.*
- *write(<int fd>, <pos>):* escreve o conteudo armazenado em <*pos*> no descritor <*fd*>.
- *wait():* bloqueia o processo pai até que um processo filho termine.

3) O seno de um ângulo (em radianos) pode ser aproximada através da série:

$$sen(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

Desenvolva um programa concorrente utilizando área de memória compartilhada que leia o valor de x (graus em radianos) e de n (número de termos da série) e calcule o seno de x. O cálculo do seno deve ser dividido entre 2 processos filhos, sendo que a soma dos termos positivos deve ser calculada por um processo e a soma dos termos negativos deve ser calculada pelo outro processo. O processo pai deve somente buscar os valores parciais obtidos pelos filhos e calcular e escrever na tela o valor total.

Para o desenvolvimento do programa utilize uma linguagem de programação real e as seguintes chamadas de sistema:

- *int fork()*: cria uma cópia do processo original. A chamada de sistema fork retorna o <*pid*> do filho para o processo pai e *0 (zero)* para o processo filho.
- *int createShm(< Número de Bytes >):* cria uma área de memória compartilhada de tamanho <*Número de bytes*> e retorna o identificador da mesma.
- <*Endereco*> *shmat*(<identificador>): retorna o endereço da área de memória compartilhada <identificador>.
- shmdt(<identificador>): usada para desanexar uma área de memória compartilhada.
- *FreeShm(< identificador>): desaloca* a área de memória compartilha <identificador>.
- *wait():* bloqueia o processo pai até que um processo filho termine.