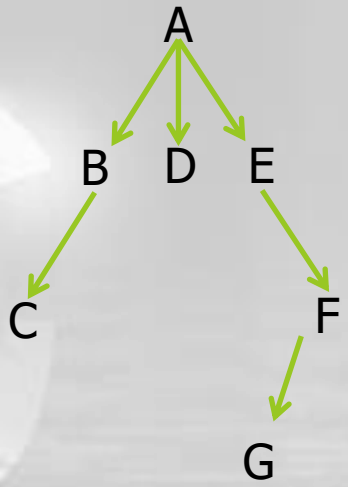
SISTEMAS OPERACIONAIS

Trabalhos

Trabalho 1 – Fork Wait

- Escreva um trecho de código que utiliza a função fork() e gera a seguinte hierarquia de processos:



- Atenção:
  - um processo deve mostrar uma mensagem se identificando ("proc-A"... "proc I")
    - quando ele acaba de ser criado
    - … e quando ele está prestes a morrer
  - cada processo gerado deve imprimir o seu PID e o PPID
  - você deve garantir que um pai sempre morre depois de seu filho!

Trabalho 1.1 – Fork Wait (extra)

- Implemente um programa C que recebe como parâmetros no comando de linha até 10 números inteiros (desordenados). O programa MAIN registra os números em um array e cria um filho . Em seguida o MAIN deve ordenar o array usando “ordenação simples” enquanto o filho deve fazer “quick sort”. Ao final da ordenação, cada processo deve exibir o tempo gasto para realizar a mesma. O processo que acabar primeiro deve matar (kill()) o seu "parente"e imprimir uma msg avisando sobre o "assassinato"(ex. "Sou o pai, matei meu filho!"). Observem que não deve ser possível que os dois processos mostrem as mensagens.
  - Dicas para resolver esse exercício no arquivo ‘roteiro1.pdf’

Trabalho 1.2 – Fork Exec (extra)

- Utilizando a estrutura de uma Shell dada pelo arquivo “shell exemplo.pdf” aumente a funcionalidade do seu Shell:
  - (i) implemente a possibilidade de mudar dinamicamente (durante runtime) o prompt usando o comando prompt> PS1=String: onde String será o novo prompt
  - (ii) implemente um comando embutido chamado “quemsoueu” que mostre detalhes sobre a identificação do usuário. Baste chamar a função system(“id”) na função “builtin”.
  - (iii) implemente o comando embutido “socpy” que permite o shell copiar um arquivo através do comando prompt> socpy destino fonte
  - (iv) implemente controlo de processos com a possibilidade de execução em foreground ou em background (adicionado o simbolo & no fim do comando)
  - (v) implemente redireccionamento do stdin e stdout (<) e (>)

Trabalho 2 - Sinais

- Escreva um programa que realiza tratamento de sinais POSIX
  - O programa deve contar quantas vezes o usuário envia o sinal SIGINT (Ctrl-C) para o processo em execução.
  - Quando o sinal receber um SIGTSTP (Ctl-Z), ele deve imprimir o número de sinais SIGINT que ele recebeu.
  - Depois de ter recebido 10 SIGINT's, o programa deve "convidar" o usuário a sair ("Really exit (Y/n)?").
    - Se o usuário não responder em 5 seg., o programa finaliza
    - Se responder 'Y' manda um sinal de termino a ele próprio.
    - Se responder 'n' reinicia contagem

Trabalho 2.1 – Sinais (extra)

- Implemente um programa que cria um novo processo e sincroniza o acesso ao arquivo "dados.txt" por meio do uso de sinais, da seguinte forma:
  - O processo pai fica à espera (não espera ocupada!) até que chegue um sinal SIGUSR1;
  - depois de receber o sinal, deve ler do arquivo "dados.txt" um número;
  - depois de lido esse número, deve remover o conteúdo desse arquivo e apresentar esse conteúdo no monitor;
  - por fim envia ao filho um sinal SIGUSR1. O processo filho é responsável por escrever um novo número no arquivo.

Trabalho 3

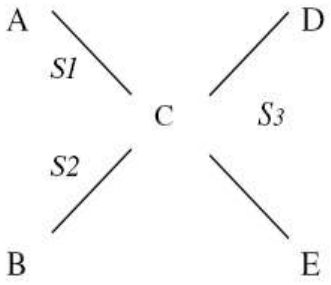
- Cálculo de pi utilizando série de gregory
  - Devem ser calculados pelo menos 1 bilhão (10^9) de termos da série
  - Use variáveis reais de dupla precisão (double) nos cálculos;
  - O programa deve dividir o espaço de cálculo uniformemente entre as N threads; cada thread efetua uma soma parcial de forma autônoma;
  - Para evitar o uso de mecanismos de sincronização, cada thread T[i] deve depositar seu resultado parcial na posição result[i] de um vetor de resultados parciais. Após o término das threads de cálculo, o programa principal soma os resultados parciais obtidos por elas e apresenta o resultado final na tela;
  - Execute as threads no seu computador pessoal e no servidor Orion
    - Acesso: faça login no linux de rede
    - Use ssh: ssh user@orion
    - user e pass são os mesmos do linux de rede
  - Em ambas as máquinas execute a solução com N = {1, 2, 4, 8 e 16} threads
    - Marque o tempo necessário para calcular Pi para cada N em cada máquina e faça um gráfico de linhas (NxTempo) apresentado os resultados obtidos em ambas as máquinas

$$\frac{\pi}{4} = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}$$

$$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots = \frac{\pi}{4}.$$

Trabalho 4

- Suponha o grafo de precedência abaixo com 5 processos. Adicione semáforos a esses processos de modo que a precedência definida acima seja alcançada
  - Ao iniciar sua execução o processo imprime na tela uma mensagem (e.g. 'Iniciando A') e espera um tempo aleatório entre 1 e 5 segundos para finalizar.
  - Ao finalizar o processo imprime uma mensagem (e.g. 'Finalizando processo 'A')




```
semaphore ...
...
Process k
        ...                     /* Comentário */
        do some work k
        ...                     /* Comentário */
end
```

Trabalho 5

- Jantar de Gauleses
  - Uma tribo gaulesa janta em comunidade a partir de uma mesa enorme com espaço para M javalis grelhados. Quando um gaulês quer comer, serve-se e retira um javali da mesa a menos que esta já esteja vazia. Nesse caso o gaulês acorda o cozinheiro e aguarda que este reponha javalis na mesa. O código seguinte representa o código que implementa o gaulês e o cozinheiro.

```
void Gaules() {
  while(true) {
    Javali j = RetiraJavali();
    ComeJavali(j);
    }
}
```

```
void Cozinheiro(){
  while(true){
    ColocaJavalis(M);
  }
}
```

  - Implemente o código das funções RetiraJavali() e ColocaJavalis() incluindo código de sincronização que previna deadlock e acorde o cozinheiro apenas quando a mesa está vazia.
  - Lembre que existem muitos gauleses e apenas um cozinheiro. Identifique regiões críticas na vida do gaules e do cozinheiro.
  - A solução deve aceitar um numero N de gauleses igual ao número de letras de seu primeiro nome e 1 único cozinheiro. Cada javali terá um nome, dado pela letra correspondente
    - Ex: dalcimar = 8 javalis
  - Cada Javali deve imprimir na tela seu nome (dado pela letra) quando come e quando acorda o cozinheiro.
    - Ex: Javali d(0) comendo
    - Ex: Javali a(1) acordou cozinheiro
  - A quantidade javalis grelhados M deve ser igual ao número dos dois primeiros dígitos de seu RA
  - A solução não deve ter nenhum comentário

Trabalho 6

- Shared Memory
  - Implemente o problema do Jantar de Gauleses usando shared memory
    - Pode usar funções da shmem ou mmap, escolha livre
    - Deve, obrigatóriamente ter um executável, código fonte separado para o produtor e outro para o consumidor
    - Os programas podem ser lançados em background (&) ou utilizando fork/exec

Trabalho 6 – Shm (extra)

- Um segmento de memória compartilhada não pode ser estendido. Escreva um programa em C que tenha, como argumento, um segmento de memória compartilhado e alguns dados. Este programa irá criar e anexar um novo e maior segmento de memória compartilhada.
- Ele copiará os dados do segmento memoria compartilhado antigo para o novo e também armazenará os dados adicionais.
- Em seguida, ele deve desanexar e desalocar o segmento de memória antigo.
- Esse programa faz, essencialmente, o mesmo que realloc() quando essa função não pode estender fisicamente um segmento de memória. Aqui, no entanto, existem alguns problemas de sincronização com que o realloc() não precisa lidar.
  - Primeiro, seu programa deve "bloquear" o segmento pela duração para que nenhum outro processo possa usá-lo.
  - Em seguida, deve certificar-se de que todos os outros processos saibam que o segmento "moveu" e "onde".
- Tente concluir este exercício usando apenas semáforos e segmentos de memória compartilhada

Trabalho 7

- Pipes
  - Implemente o problema do Jantar de Gauleses usando pipes
    - Usar pipes, não usar named pipes
    - Deve, obrigatóriamente ter um executável, código fonte separado para o produtor e outro para o consumidor
    - Os programas podem ser lançados utilizando fork/exec

Trabalho 8

- OpenMP
  - Implemente o problema do Jantar de Gauleses usando OpenMP
    - Deve, obrigatóriamente ter um único executável, código fonte o produtor e para o consumidor no mesmo arquivo
    - O número de produtores e consumidores pode ser escolhido automaticamente pelo OpenMP

Trabalho 8 (extra)

- Sockets
  - Implemente o problema do Jantar de Gauleses usando sockets
    - Utilize sockets Unix ou na interface 127.0.0.1
    - Deve, obrigatóriamente ter um executável, código fonte separado para o produtor e outro para o consumidor
    - Para alcançar sincronização os o processo Gaules envia mensagens vazias para o Cozinheiro para cada Javali Grelhado comido. O Cozinheiro então acordará quando M mensagens vazias tiverem chegado (M javalis grelhados), cozinhará os Javalis as os enviará para os Gauleses. Caso não haja M mensagens vazias Cozinheiro dorme.

Trabalho 9

- MPI
  - Implemente o problema do Jantar de Gauleses usando MPI
    - Deve, obrigatóriamente ter um único executável, código fonte o produtor e para o consumidor no mesmo arquivo

Trabalho 10 – MPI+OpenMP

- Implemente o problema do Jantar de Gauleses usando MPI e OpenMP
  - Deve, obrigatóriamente ter um único executável, código fonte o produtor e para o consumidor no mesmo arquivo
  - O buffer tem que ser 5 (utilize mensagens vazias)
    - Dicas:
    - https://www.sharcnet.ca/~jemmyhu/tutorials/MPI+OpenMP_2016.pdf
    - http://www.slac.stanford.edu/comp/unix/farm/mpi_and_openmp.html
    - https://docs.loni.org/wiki/Introduction_to_Programming_Hybrid_Applications_Using_OpenMP_and_MPI
    - https://www.ee.ryerson.ca/~courses/ee8218/mpi_openmp.pdf
    - https://doc.itc.rwth-aachen.de/download/attachments/3474275/OpenMP_and_MPI_for_Dummies.pdf?version=1&modificationDate=1391432738000&api=v2