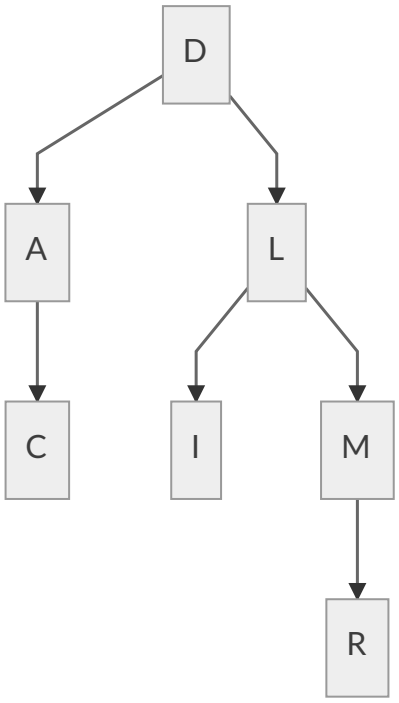
Trabalhos SO

Trabalho 1 – Fork Wait

Escreva um trecho de código que utiliza a função fork() e gera uma árvore de busca binária (ou quase isso) com seu primeiro nome. Para tal siga as seguintes regras:

1. Primeira letra será a raiz da árvore
2. Cada letra seguinte será inserida a direita, se a letra for maior que a raiz, ou a esquerda, se a letra for menor que a raiz.
3. Esse procedimento (verificar a raiz e inserir a direita ou a esquerda) deve ser realizado recursivamente.
   Ex: dalcimar



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!

Para a árvore acima a saída poderá ser:

```
proc-D, pid 2454, ppid 1354, acaba de ser criado
proc-A, pid 2455, ppid 2454, acaba de ser criado
proc-L, pid 2456, ppid 2454, acaba de ser criado
proc-C, pid 2457, ppid 2455, acaba de ser criado
proc-C, pid 2457, morreu
proc-A, pid 2455, morreu
proc-I, pid 2458, ppid 2456, acaba de ser criado
proc-I, pid 2458, morreu
proc-M, pid 2459, ppid 2456, acaba de ser criado
proc-R, pid 2460, ppid 2459, acaba de ser criado
proc-R, pid 2460, morreu
proc-M, pid 2459, morreu
proc-L, pid 2456, morreu
proc-D, pid 2454, morreu
```

Trabalho 2 - Sinais

Escolher 1 dos trabalhos listados abaixo

Trabalho 2.1

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 2.2

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.

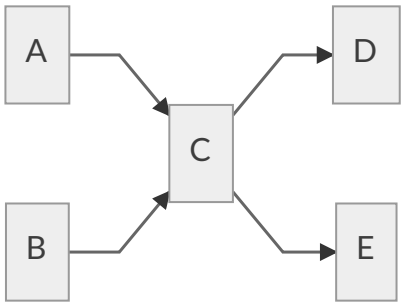
Trabalho 3 - Threads

Escolha duas séries quaisquer para aproximar o número pi do site Pi series 1 ou Pi series 2 ou outro site qualquer. Veja exemplos sequenciais em Pi sequencial. Implemente a versão paralela dessas séries, utilizando pthreads, sequindo os seguintes requisitos:

- Devem ser calculados pelo menos 1 bilhão (10^9) de termos de cada série.
- Use variáveis reais de dupla precisão (double) nos cálculos;
- O programa deve dividir o espaço de cálculo uniformemente entre as N threads
  - e.x. 1bilhão de termos com 2 threads = 500milhões de termos em cada thread;
  - 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
- Execute as soluções com N = {1, 2, 4, 8 e 16} threads
- Marque o tempo necessário para calcular Pi para cada N e faça um gráfico de linhas (NxTempo) apresentado os resultados obtidos
- Compare o resultado das duas soluções (series) escolhidas, indicando qual série é mais eficiente em termos de **tempo** e **qualidade da solução** (i.e. valor mais exato de pi)

Trabalho 4 - Grafo de precedência

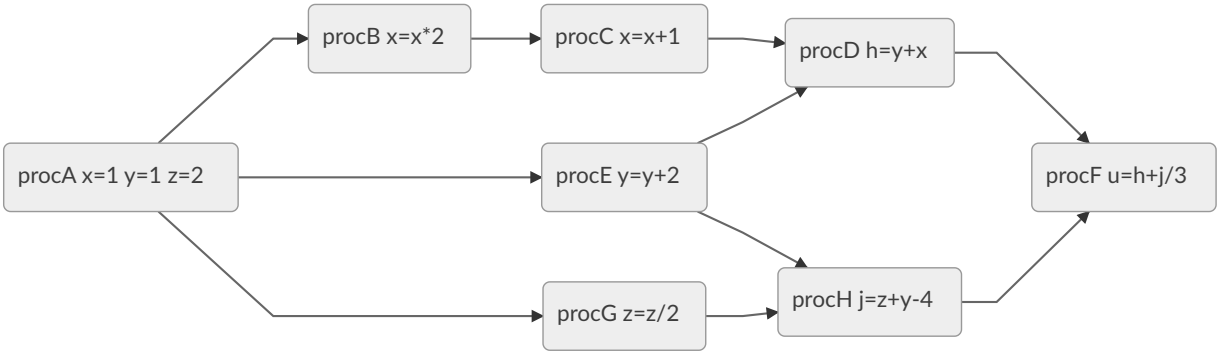
Suponha o grafo de precedência abaixo com 5 processos.



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’)

Trabalho 5 - Grafo de precedência

Considere o seguinte grafo de precedência:



Que será executado por três processos, conforme código abaixo:

- P1: begin A; E; F; end;
- P2: begin B; C; D; end;
- P3: begin G; H; end;

Adicione semáforos a este programa, e as respectivas chamadas às suas operações, de modo que a precedência definida acima seja alcançada.

Obdeça as equações obtendo valor final de u dado as entradas de x, y e z

Trabalho 6 - Jantar de Gauleses - Threads

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 gaules terá um nome, dado pela letra correspondente
  – Ex: dalcimar = 8 gauleses
- Cada gaules deve imprimir na tela seu nome (dado pela letra) quando come e quando acorda o cozinheiro.
  – Ex: Gaules d(0) comendo
  – Ex: Gaules 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 7 - Jantar de Gauleses - Shared Memory

Implemente o problema do Jantar de Gauleses usando shared memory

- Pode usar funções da shmem (system V) ou mmap (POSIX), escolha livre. Recomendo padrão POSIX
- 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 8 - Jantar de Gauleses - Pipes

Implemente o problema do Jantar de Gauleses usando pipes

- Usar pipes ou 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
- Controlar o tamanho do buffer usando pipes é problemático, já que o pipe tem um tamanho fixo dado pelo sistema operacional. Sugestões de solução para 'emular' um buffer usando pipes
    - não fazer nada (terá um pequeno desconto na loja)
    - descobrir o tamanho do pipe e mandar dados de tamanho correto. Ex, pipe 100kb total, buffer de 5 elementos, cada 'write' ou 'read' no pipe feito com 20kb (100/5)
    - utilizar um esquema de passagem de mensagens (consumidor envia 5 mensagens vazias para produtor)
- Desenvolva uma estratégia para comunicar 1 produtor (cozinheiro) e N consumidores (gauleses). Sugestão utilizar threads no processo gaules.

Trabalho 9 - Jantar de Gauleses - Sockets

Implemente o problema do Jantar de Gauleses usando Sockets

- Utilize sockets Unix (sugerido) ou na interface 127.0.0.1
- Deve, obrigatóriamente ter um executável, código fonte separado para o produtor e outro para o consumidor
- O programa deve permitir um 1 produtor e N consumidores
- O tempo do produtor e do consumidor deve ser aleatório
- Para alcançar sincronização os o processos Gaules enviam 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.

- O controle do consumidor que enviou a mensagem pode ser feito usando a própria mensagem (cabeçalho) ou pela estrutura de resposta do método conect ou utilizando uma arquitetura do tipo mailbox.

Trabalho 11 - Jantar de Gauleses - 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

Trabalho 12 - Jantar de Gauleses - Passagem de mensagem

Implemente o problema do Jantar de Gauleses usando MPI (família mpi_send) ou POSIX (famílio mk_send) escolha livre.

- Recomendo padrão POSIX, pois foi a ensinada nos vídeos e está nativamente instalada. Para quem quiser usar MPI tem slides, e vc irá precisar instalar o pacote.
- Deve, obrigatóriamente ter um único executável, código fonte o produtor e para o consumidor no mesmo arquivo

Trabalho 13 - Jantar de Gauleses - MPI + 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: link1 link2 link3 link4 link5