**Linguagem c**

**Printf();**  (comando de saida (escreva))

**Sleep();** (Comando que faz a mensagem ficar ativa por determinado tempo)

**gcc (nome do programa com o “.c”) –o (nome do programa sem o “.c”)** (comando do linux que compila depois de editar o código)

**./(nome do programa sem o “.c”)** (Comando que executa o código)

**(sleep1 & exec/bin/sleep15)** (Esse comando cria um processo que dura 1 segundo e gera um segundo processo (filho) que dura 15 segundos, ou seja, o processo filho não sabe que o processo pai foi terminado antes dele. Isso criará um processo zumbi por 15 segundos!)

**Escalonamentos**

**First-in-First-out (FIFO)**

First-in – Processo que entra

First-out – Processo que sai

Média - 8 Unidades de tempo

Mudar a ordem de processo ganha tempo (ou perde)

**Shorter-job-first (SJF)** - Mais curto primeiro

Mais rápido doque o FIFO

**Corporativo** – O escalonamento corporativo no Linux é como dar prioridade aos aplicativos mais importantes para uma empresa. Se um sistema, como um banco de dados, é crucial, mais recursos como CPU e memória são dedicados a ele, garantindo que funcione sem problemas, mesmo sob carga pesada. É como garantir que as tarefas mais importantes recebam mais atenção para manter o negócio funcionando sem problemas.

**Por prioridade** – o processo que for mais importante é executado primeiro

**Múltiplas filas** – Faz um escalonamento por meio de tipos de processos, como o do menos importante, do médio e do mais importante, quando terminar a execução da fila de um, a outra fila começa

**Circular –** funciona da seguinté maneira: imagine que cada processo esteja encima de uma esteira, onde um braço robótico pega esse processo e executa-o com um tempo pré estabelecido pelo Quantum, quando esse tempo termina o processo volta a fila (mesmo se não concluído) e volta ao final da fila encima da esteira pois será a vez de outro processo, quando chega novamente a vez desse primeiro processo ele continua da onde parou da primeira vez e é executado novamente até que seu tempo cheguei ao limite de novo, se o programa terminar sua execução antes do tempo pré estabelecido a mão robótica pega ele e o tira da linha (esteira) e o coloca numa sala de espera, caso ele não tenha terminado sua execução e o tempo tenha chegado ao limite, todo aquele ciclo da primeira vez é feito novamente até que o mesmo terminei sua execução

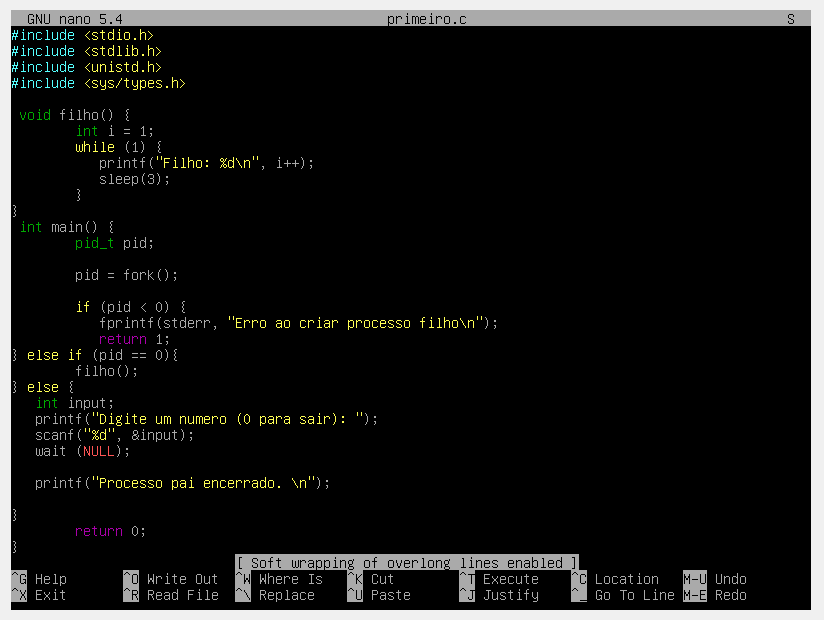
**Em tempo real;** No escalonamento em tempo real, cada tarefa crítica recebe total atenção até ser concluída ou até o tempo acabar. Quando uma termina, a próxima começa, garantindo que todas as tarefas importantes sejam realizadas dentro dos prazos estabelecidos.

**Exemplo:** imagine que você tem várias tarefas importantes para fazer em casa, como cozinhar o jantar, lavar a roupa e limpar a casa. Cada tarefa tem um prazo específico para ser concluída: o jantar precisa estar pronto até as 19h, a roupa precisa ser lavada até as 20h e a casa precisa estar limpa até as 21h.

Nesse caso, você prioriza suas tarefas com base nesses prazos. Por exemplo, se forem 18h e você ainda não começou a cozinhar o jantar, você vai focar nessa tarefa primeiro, porque é a mais urgente. Você deixa a lavagem da roupa e a limpeza da casa para depois, garantindo que o jantar fique pronto a tempo.

**Exercício de criação do programa que cria dois processos (pai e filho)**

**Código em C**



**Relatório: Desenvolvimento e Execução de um Programa em C com Processos Pai e Filho**

**Introdução:** O objetivo deste relatório é descrever o desenvolvimento e a execução de um programa em C que demonstra o uso de processos pai e filho. No programa desenvolvido, o processo pai solicita entradas de teclado ao usuário, enquanto o processo filho exibe uma sequência numérica com incremento a cada 3 segundos.

**Desenvolvimento:** O programa foi desenvolvido na linguagem C, utilizando as bibliotecas padrão stdio.h, stdlib.h, unistd.h e sys/types.h. O código foi estruturado em duas principais funções: filho() e main().

A função filho() é responsável por exibir a sequência numérica com um incremento de 3 segundos entre cada número. Ela utiliza um loop infinito para continuar exibindo os números indefinidamente.

Na função main(), o programa cria um novo processo filho utilizando a função fork(). Se a chamada a fork() for bem-sucedida, o processo pai recebe um valor de retorno positivo que corresponde ao PID (Identificador de Processo) do processo filho. O processo filho recebe um valor de retorno zero. Com base nesse valor de retorno, o código decide qual parte do programa deve ser executada por cada processo.

No processo filho, a função filho() é chamada para iniciar a exibição da sequência numérica.

No processo pai, o programa solicita um número ao usuário através da função scanf(). Após receber a entrada do usuário, o processo pai aguarda a conclusão do processo filho antes de encerrar, para garantir uma terminação ordenada de ambos os processos. Isso é feito utilizando a função wait() com NULL como argumento, o que faz com que o pai espere qualquer processo filho terminar.

**Funcionamento**: Após a execução do programa, o processo pai solicitará ao usuário que digite um número. Enquanto isso, o processo filho começará a exibir uma sequência numérica com um incremento de 3 segundos entre cada número. O programa continuará em execução até que o usuário digite 0 como entrada no processo pai. Quando isso acontecer, o processo pai encerrará e, consequentemente, o processo filho também terminará.

**Conclusão:** Este relatório descreveu o desenvolvimento e a execução de um programa em C que utiliza processos pai e filho para realizar tarefas distintas. A criação de processos é uma técnica importante em programação concorrente e sistemas operacionais, e este programa serve como um exemplo prático para entender como os processos podem ser utilizados para realizar tarefas simultaneamente em um sistema operacional multitarefa.