quarta-feira, 22 de junho de 2016 23:06

SISTEMAS TEMPO COMPARTILHADO:
Permitem que diversos programas sejam executados a partir da divisão do tempo do processador em pequenos intervalos, denominados fatia de tempo (time-slice). Caso a fatia de tempo não seja suficiente para a conclusão da tarefa, ela é interrompido pelo cpu e substituído por um outro sequencialmente, enquanto fica aguardando por uma nova fatia de tempo. Os tempos dedicados para cada tarefa são pequenos o suficiente para dar a ilusão de que as tarefas estão sendo executadas simultaneamente
SISTEMAS REAL TIME: sem fatia do tempo, o processo executa ate outro mais prioritario aparecer e interromper a execucao.

MULTIPROGRAMACAO: nele ocorre ocorre o compartilhamento da memória e do processador. gerenciar o acesso concorrente.

CRITERIOS DE ESCALONAMENTO: utulizacao da CPU sempre 100%, throughtpu(produtividade), turaround(tempo de processamento), tempo de espera

round robin - projetado especialmente para sistemas de tempo compartilhado
com preempsao

PCB: bloco de contole do processo
Guarda estado do processo, contador pra proxima instrucao e registradores
Troca de contexto/PCB: Retomar a execução no ponto onde parou no programa do usuário e recomeçar
Contem id, estado, registradores(estrutura de dados), areas de memoria e contadores(endereco da proxima instrucao),

- Suspende execucao de a
- Transpere controle para o escalonador de cpu
- Salva contexto de a em pcba
- Carrega o controle de B apartir do pcbb
- Retorma execucao de instrucoes de B

THREAD: cada thread tem o registradores, espaco de pilha na memoria e recursos. o tempo de processamento pelo escalonamento é menor que o de processos, já que é mais rapido chegam mais perto na visao do usuario de um processaento simultaneo em multiplos cpus as thread são realizadas de forma simultanea. sao independentes do processo e podem ser escolanadas na ordem que o escalar determinar prioridade

Kernel/supervisor = sudo usuario = ok
Instrucoes privilegiadas precisam de system calls(altera modo de acesso) para impedir problemas de seguranca e garantir integridade do sistema. Não privilegiadas nao oferecem risco.

EXCLUSAO MUTUA: tecnica para evitar que dois processos ou threads tenham acesso simultaneo a um recurso compartilhado
SEMAFORO: trava o recurso deixa outros processos esperarem e libera após o uso para o proximo da fila. Causam deadlocks em q dois processos tem o mesmo semaforo e ficam esperando um o outro liberar e nunca conseguem ter exito esperando regiao critica

```
typedef int semaphore; /* semaphores are a special kind of int */
semaphore mutex = 1; /* controls access to critical region */
semaphore empty = N; /* counts empty buffer slots */
semaphore full = 0; /* counts full buffer slots
```

```
void producer(void)
while(true)
produce_item(&item); /* generate next item */
down(&empty); /* decrement empty count */
down(&mutex); /* enter critical region */
enter_item(item); /* put item in buffer */
up(&mutex); /* leave critical region */
up(&full); /* increment count of full slots */
```

```
void consumer(void)
{
int item;
while (TRUE) { /* repeat forever */
down(&full); /* decrement count of full slots */
down(&mutex); /* enter critical region */
remove_item(&item); /* take item from buffer */
up(&mutex); /* leave critical region */
up(&empty); /* increment count of full slots */
consume_item(item); /* use item */
```

DEADLOCK espera uso exclusivo. Espera circular espera recurso alocado. Quando p/t precisa de um recurso alocado precisando de outro recurso.

SECAO CRITICA: instrucoes não podem ser usadas de forma simultanea necessario sincronizar para acesso exclusivo. É a parte do programa, onde o processamento pode levar a ocorrência de condições de corrida

CONDICAO DE CORRIDA: problema em concorrencia em que threads dependem de um espaco compartilhado que nao pode ser violado e o resultado depende da ordem do escalonamento.

CRIACAO: neste estado o processo está sendo alocado na memória, sendo criado no sistema. seus recursos necessários à execução do processo são reservados.
PRONTO: é o estado onde os processos, depois de criados ou quando retornam do tratamento de uma interrupção(preempcao), permanecem aguardando a liberação da CPU para que possam iniciar ou continuar seu processamento(serem despachados pelo escalonador) fila dependendo da prioridade do escalonador
EXECUCAO: é onde o processo efetivamente utiliza a CPU. Ele permanece no processador até que seja interrompido(preemptado) ou termine sua execução.
ESPERA/BLOCK: neste estado estão todos os processos que sofreram algum tipo de interrupção de E/S, onde permanecem até que a intervenção seja resolvida.

-

Latência do Despachante: tempo gasto para parar um programa e começar outro

Processo I/O Bound: tempo de processamento depende mais do tempo de processamento de operações de E/S do que o tempo de processamento de CPU.
Processo CPU Bound: tempo de processamento depende mais da CPU. tem em geral algumas fases em que a CPU é usada por um período muito longo de tempo.

Procedimento de criação:
□cria o segmento de memória compartilhada (shmget)
□anexa o segmento para poder (shmat)
□após o uso desanexa (shmdt)
□finalmente um processo desaloca

MEMORIA COMPARTILHADA: permite 2 ou mais processos compartilhar uma regiao de memoria
. forma mais simples e rapida de ipc. necessario sincronizar o acesso a regiao por multiplos processos com semaforos
Procedimento de uso:
□obtém o identificador da memória compartilhada (shmget)
□ cada processo que deseja utilizar o segmento anexa o mesmo (shmat)
□após o uso desanexa (shmdt)
□finalmente um processo desaloca

PIPE
fd[1] escreve em -> fd[0] le int fd[2] pid_t pid pipe(fd);

fd[1] <-[pai]-> close write(fd[1], "oi", 2)
close <-[fio]-> fd[0] n = read(fd[0], line, maxline)

Filho close1 dup2 (fd[0], 0) close 0 execlp(x,x,(char*)0)
Pai 0 1 1

Fila circular: Ponteiros se movem em direcao ao inicio do array, evita desperdico de tempo ocasionado pelos deslocamentos dos

Fila circular: Ponteiros se movem em direção ao início do array, evita desperdício de tempo ocasionado pelos deslocamentos dos elementos da filas às primeiras posições. não há preocupação para quando o ultimo elemento da fila atinge a posição máxima do vetor pois a última posição é do lado da primeira.

OPERACOES ATOMICAS: São operações que não podem ser interrompidas. Não é possível ver as "partes" de uma operação atômica, mas apenas seu efeito final.

EXCLUSAO MUTUA: É a garantia que dois ou mais processos não acessem a região crítica ao mesmo tempo.

- Dois os mais processos não podem estar simultaneamente dentro de suas regiões críticas.
- Nenhum processo que esteja rodando for a da RC pode bloquear a execução de outro processo.
- Nenhum processo pode ser obrigado a esperar indefinidamente para entrar em sua RC.

Ser justo :dar acesso a todos eventualmente.
☐ Eficiente : não utilizar quantidades substanciais de recursos quando estiver esperando. Em particular, evitar “espera ocupada”.
☐ Simples : deve ser fácil de utilizar.

Apenas um processo (pessoa) pode fazer alguma coisa em determinado momento. Exemplo : apenas uma pessoa pode sair para comprar cerveja em qualquer momento. ☐ Temos que tornar o acesso a região crítica uma operação atômica (comprar cerveja), ou seja, apenas um processo (pessoa) pode executar de cada vez.

☐ Os mecanismos de sincronização podem ser classificados de duas formas: ☐ Com espera ocupada (busy waiting) : O processo ao entrar na região crítica inibe as interrupções, garantindo assim que o processo não será interrompido por exceder o tempo de processamento cedido a ele. ☐

Inibicao de interrupcoes
Ao sair da região crítica o processo habilita as interrupções.

O processo que deseja entrar na RC muda para 1 esta variável, e ao sair coloca o valor 0 na variável de travamento.

```
while (TRUE) {
        while (var_trava !=0);
        var_trava = 1;
        regiao_crítica();
        var_trava = 0;
}
```

1-1 maior concorrencia
X-1 processo é bloqueado se um dos threads usar uma chamada bloqueante
X-x evita criacao excessiva de hreads