1Sistema de Ficheiros: Parte do SO responsável por gerir o **acesso** à **memoria em massa**.

Ficheiro: **unidade lógica** de armazenamento em memoria em massa. (A leitura e a escrita são sempre feitas consoante um script de um ficheiro.)

Sistema operativo: Tem o papel de implementar este tipo de dados abstratos ao programador (ficheiros), fornecendo um conjunto de operações (**system calls**) que permitem estabelecer uma comunicação simples e segura com a memoria em massa.

Tipos de ficheiros:

Ficheiros ordinários- ficheiros cujo conteúdo é da responsabilidade do utilizador (ex word)

Diretório- ficheiro usado para rastrear, organizar, localizar outros ficheiros ou diretórios.

Atalhos -ficheiro que contem referencia para outro

Socket- ficheiro usado para comunicações dentro de processos ou dentro da máquina.

Inodes – corresponde ao cartão de identificação do ficheiro.

2Introdução a sistemas operativos

Sistema operativo: Programa base (suporte) que é executado por um sistema computacional. Dá vida ao hardware proporcionando um ambiente de **interação abstrato**. Funciona como uma interface entre a máquina e programas de aplicação. Aloca recursos partilhados do sistema, de forma dinâmica, para os programas em execução.
Objetivos SO:

- conveniência na forma como o computador é usado;
- eficiência no uso de recursos do computador;
- capacidade de evoluir para permitir o desenvolvimento de novas funções do sistema;

Multiprocessing- Paralelismo- capacidade de um sistema computacional correr simultaneamente dois ou mais programas, para isso é necessário um processador para cada execução em simultâneo.

Multiprograming- Capacidade de o sistema computacional criar uma ilusão de aparentemente ser capaz de correr simultaneamente mais programas do que o nº existente de processadores.

Concorrência: vários programas supostamente a serem executados no mesmo processador

Programa: conjunto de instruções que descrevem como uma tarefa é realizada pelo computador.
Processo: Aparece como meio de controlar atividade de vários programas executados num sistema multiprogramado. Um processo não é um programa, mas sim uma **entrada que representa um programa do computador a ser executado**.

Nota: Diferentes processos podem correr no mesmo programa.

Gestão de memoria: A partilha de memória deve ser possível de uma forma controlada. O **armazenamento a longo prazo** também deve ser possível uma vez que muitas aplicações de

programas precisam de armazenar informação por longos períodos de tempo, depois de o computador ser desligado. Isto é possível com:

- Memória virtual que dissocia a memoria vista por um processo e a memoria real.
- Sistema de ficheiros que introduzem o conceito de ficheiro como meio à memoria de longo prazo.

A alocação de recurso e a política de agendamento deve fornecer:

- Justiça: dar acesso quase igual a todos os processos
- Responsabilidade diferencial: agir considerando o conjunto total de requisitos.
- Eficiência: tentativa de maximizar o rendimento, minimizar o tempo de resposta, e no caso de sistemas com partilha de tempo, acomodar tantos usuários quanto possível (se não houver conflitos).

3 Processos

Há um processador virtual por cada processo existente, mas o numero de processadores virtuais ativos tem de ser menor ou igual que o numero de processadores reais.

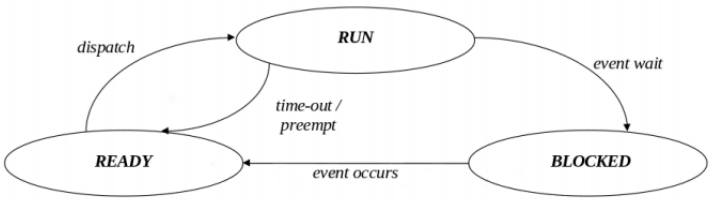
Estados do processo:

Curto prazo:

Estados:

- Run- o processo está na posse de um processador, ou seja, está a correr
- Blocked- o processo está à espera que corra um evento externo(acesso a um recurso, termino de uma operação in/out…)
- Ready- O processo está pronto a correr, mas está a espera da disponibilidade do processador para começar/ resumir a execução

Short-term state diagram



Dispatch- Um dos processos prontos a correr é selecionado e é-lhe dado um processador.
Event ocurs- Um evento externo ocorreu e o processo fica a aguardar por um processador.
Timedout- O time quantum atribuído ao processo chegou ao fim, então o processo é removido do processador.
Preempt- Chegou um processo de maior prioridade pelo que o acesso ao processador é retirado ao atual para dar ao de maior prioridade.
Event wait – O processo em execução é impedido de continuar à espera que u evento externo ocorra.

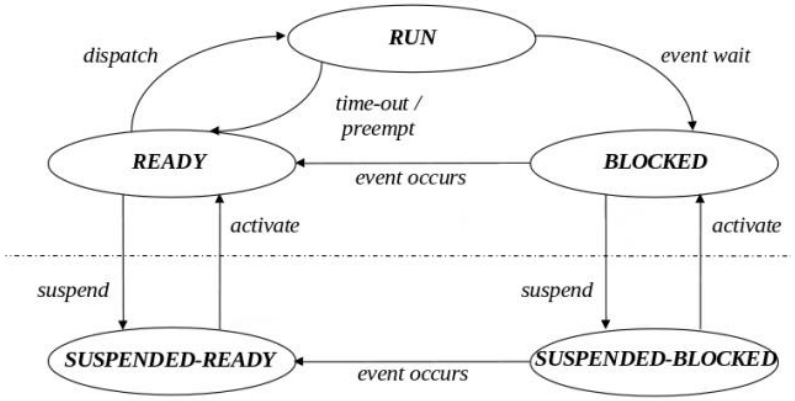
Medio prazo:

A memoria principal é finita, o que limita o número de processos coexistentes. Uma forma de ultrapassar isto é usar uma área na memória secundária para estender a memória principal – SWAP AREA. Esta pode ser uma partição do disco ou um ficheiro. Um processo que não esteja a correr pode sofrer swapped out de maneira a libertar a memoria principal para outros processos. Assim que a memoria principal ficar disponível este processo sofre swapped in.

Estados:

- Suspended-ready- o processo está pronto, mas sofre swapped out;
- Suspended- blocked o processo está bloqueado e sofre swapped out;

State diagram, including short- and medium-term states



Suspended- O processo sofre swapped out;

Activate- O processo sofre swapped in;

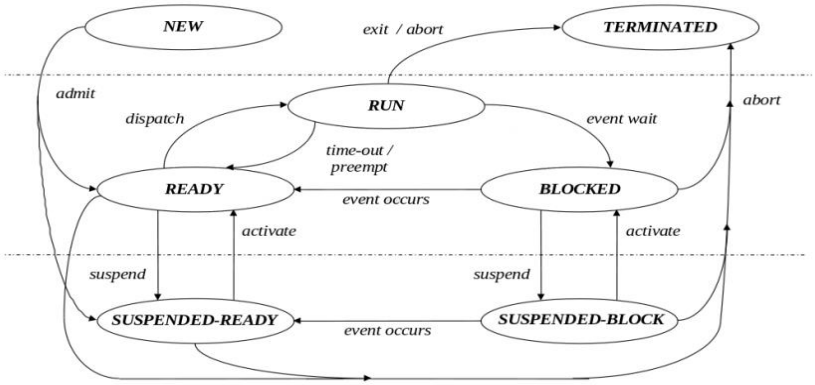
Longo prazo:

Os diagramas atrás assumem que os processos são eternos, mas à parte de alguns sistemas de processos então não é verdade. Os processos são criados, existem por algum tempo e eventualmente terminam.

Estados:

- New- o processo foi criado, mas não foi admitido para a pool de processos executáveis (a estrutura de dados do processo já foi inicializada).
- Terminated- o processo foi libertado de pool, mas algumas ações ainda são necessárias antes do processo ser descartado.

Global state diagram



Admit- o processo é admitido pelo sistema operativo para a pool de processos executáveis
Exit- o processo a correr indica ao sistema operativo que está concluído.
Abort- o processo é forçado a terminar por causa de um erro fatal ou pq um programa autorizado assim o exigiu.

Processamento de uma exceção normal

1) Salva Pc e PCW do processo em execução na stack do sistema (PC = rotina de tratamento de exceções)
2) Salva os registos
3) **Processa**
4) E restaura os registos
5) Restaura PC e PCW

Processamento de uma troca de processos

1) Guardar os dados do processo a correr (espaço de endereçamento, registos, I/o data..)
2) Atualiza o estado do processo (também na Process control table)
3) Seleciona um processo que esteja ready
4) Atualiza o estado do processo selecionado
5) Restaura dados do processo selecionado na PCT

4 Threads

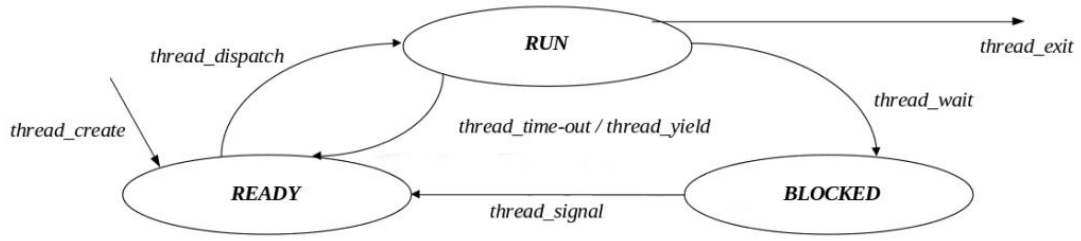
Podem ser vistas como processos de baixo peso. Num SO normal, um processo inclui:

- Espaço de endereçamento(código e dados);
- Conjunto de canais de comunicação I/O;
- Uma única thread de controlo, que incorpora os registos do processador e uma stack.

Neste modelo, thread aparece como uma componente de execução dentro de um processo.
Multithreading: diversas threads independentes podem coexistir no mesmo processo partilhando o mesmo EE e contexto IO.

Estados das threads:

Apenas os estados relativos à gestão do processo são considerados (estados de curto prazo), o resto ou estão relacionados com processos e não com threads (estados suspended) ou por a gestão de um ambiente multiprogramável ser basicamente a restrição do numero de threads que podem existir num processo (estados new e terminated).



22/46

Processos em unix:
Fork: cria uma replica do processo. No instante depois da instrução, o EE dos 2 processos são iguais. Algumas Variaveis dos processos diferem (PID,PPID)
PID pai = PID filho;
PID filho=0;

Clone: cria um novo processo que pode partilhar elementos com o pai.

5 Comunicação entre processos

```
/* control data structure */
#define  R    2          /* process id = 0, 1 */
shared bool is_in[R] = {false, false};
void enter_critical_section(unsigned int own_pid)
{
    unsigned int other_pid = 1 - own_pid;
    while (is_in[other_pid]);
    is_in[own_pid] = true;
}
void leave_critical_section(unsigned int own_pid)
{
    is_in[own_pid] = false;
}
```

Exemplo de código que não leva a exclusão mútua por testar primeiro a variável de controlo do outro e depois mudar a sua.

Secções críticas devem ser executadas em **exclusão mútua**. No entanto exclusão mútua no acesso a um recurso ou área partilhada pode resultar em:
Deadlock: quando 2 ou mais processos estão a espera eternamente de eventos que nunca vão acontecer para aceder à sua secção crítica. As operações são bloqueadas.

```
/* control data structure */
#define  R    2          /* process pid = 0, 1 */
shared bool want_enter[R] = {false, false};
void enter_critical_section (unsigned int own_pid)
{
  unsigned int other_pid = 1 - own_pid;
  want_enter[own_pid] = true;
  while (want_enter[other_pid]);
}
void leave_critical_section (unsigned int own_pid)
{
  want_enter[own_pid] = false;
}
```

Exemplo de código que tem exclusão mútua, mas conduz a deadlock, os 2 podem ficar a true e ficar a espera e nenhum cede.

Starvation: quando um ou mais processos competem por acesso a uma secção critica e este acesso é continuamente adiado, devido a um conjunto de circunstâncias nas quais novos processos que têm maior prioridade chegam continuamente.

Para aceder a uma **secção crítica**:
-exclusão mútua: o acesso às secções críticas associadas ao mesmo recurso, só podem ser permitidas a um processo de cada vez.
-Independente do número de processos intermitentes ou da sua velocidade de execução relativa.
-Qualquer processo que esteja fora da região critica não deve impedir que outro que queria entrar de o fazer.
-Sem starvation: deve impedir que um processo fique indefinidamente à espera.
O processo que está dentro, deve estar por um tempo finito.

Tipos de soluções:
Localização em memoria: funciona como uma flag binária e é usada para controlar o acesso a uma secção crítica.
Soluções de software: soluções baseadas nas instruções típicas de acesso à memoria: read e write são feitos por instruções diferentes (independentes), entre eles podem ocorrer interruções;
Algoritmo de Peterson:
Enquanto houver um outro processo com maior prioridade que o own fica preso no while. Não há starvation porque é um for e tem um número limitado de ciclos.

```
#define  R    ...     /* process id = 0, 1, ..., R-1 */
shared int level[R] = {-1, -1, ... , -1};
shared int last[R-1];
void enter_critical_section(uint own_pid)
{
   for (uint i = 0; i < R-1; i++)
   {
      level[own_pid] = i;
      last[i] = own_pid;
      do
      {
         test = false;
         for (uint j = 0; j < R; j++)
            if (j != own_pid)
               test = test || (level[j] >= i);
      } while (test && (last[i] == own_pid));
   }
}
void leave_critical_section(int own_pid)
{
   level[own_pid] = -1;
}
```

Soluções de Hardware: soluções baseadas em instruções especiais para aceder à memoria. Estas permitem a leitura e depois escrita na memoria de forma atómica.

- No caso de um sistema computacional uniprocessador, a comutação entre processos é sempre causada por um device externo. Então o acesso em exclusão mútua pode ser implementado desativando as interrupções. Deve ser feito apenas a nível do kernel. Há o requisito de bloquear o processo enquanto este está à espera de entrar na sua secção crítica.
- No caso de um sistema multiprocessador isto já não tem efeito. Tem de se utilizar instruções especiais para contruir as primitivas de entrada e saída da secção critica.
    - P.ex. Função test_and_set que faz a verificação de se a alteração do valor de uma variável foi feita por mim(processo) ou não de forma a que é esse processo que ganha acesso. Isto à custa de busy waiting.

  Busy waiting: A primitiva lock está no estado ativo (a usar o CPU) enquanto espera.
  SpinLock: O processo girá à volta da variável enquanto espera por acesso. Em sistemas multiprocessadores está solução acaba por não ser má uma vez que evita também comutação entre processos, que também custa.

6 Semaphores

```
typedef struct
{
   unsigned int val;      /* can not be negative */
   PROCESS *queue;        /* queue of waiting blocked processes */
} SEMAPHORE;
```

Semáforos são um mecanismo de sincronização, definido por uma datatype e 2 operações atómicas(ambas têm disable das interrupções):
Down: bloqueia processos se val for zero, caso contrário decrementa val. Espera o tempo necessário para conseguir entrar.

Up: se a queue não esta vazia, acorda um dos processos a espera (isto acontece porque um processo fez down e não conseguiu prosseguir porque o acesso estava fechado). Se estiver vazia, incrementa val.
Nota: O up pode vir primeiro que o down
A implementação é típica em sistemas de uniprocessador uma vez que a o disable das interrupções não impede que alterações sejam feitas noutros processadores.
Vantagens:

- Suportado a nível do sistema operativo;
- Gerais: primitivas de muito baixo nível;

Desvantagens:

- Exige conhecimento muito especializado sobre **concorrência;**
- É preciso primeiro garantir condições para avançar e só depois é que avança (**bottom-up**). Se as condições necessárias não forem satisfeitas, os processos são bloqueados **antes de entrarem na sua secção critica**.
- São usados para garantir exclusão mútua e sincronização de processos simultaneamente.

7 Monitores

São primitivas de baixo nível onde se aplica uma perspetiva **top-down** onde os processos primeiro entram nas suas secções críticas e depois bloqueiam se as condições não forem satisfeitas.
Trata a exclusão mútua e sincronização de forma separada graças à solução de introduzir a uma construção concorrente a nível da linguagem de programação.
As threads são primitivas a nível do sistema operativo que permitem contruir monitores.
A sincronização entre threads é possível através de variáveis de condição. Nelas podem ser feitas 2 operações:
Wait- a thread é bloqueada e posta fora do monitor retira-a da exclusão mútua. Obriga sempre ao bloqueio do código. É necessário associar uma condição à variável de condição.
Signal (Broadcast)- se há threads bloqueadas uma é acordada. É efémore, não há uma variável de de recurso que guarde o seu valor.

Hoare monitor:
A thread que manda o signal vai se colocada num waiting stack e vai ser corrido a thread para a qual o signal foi enviado. Quando o wait acabar, a thread que enviou o signal volta. A waiting stack existe pq se pode lançar vários signals

Brinch Hansen monitor:
Da mão do processo. Sai do CPU e não volta, ao contrário do Hoare.

Lampson/Redell monitor
A thread "principal" envia o signal que notifica outra thread 2 que esta poderá ter a possibilidade de concorrer pelo CPU assim que a thread que mandou o signal acabar o que tem a fazer.

8 Filas de mensagens:

Os processos podem comunicar por troca de mensagens.
São um mecanismo geral de comunicação que inclui comunicação e sincronização, valido para sistemas uni e multiprocessing. É necessário um link de comunicação.

Comunicação síncrona direta simétrica: quem envia diz para quem envia, quem recebe diz de quem recebe. Há apenas um canal de comunicação entre os 2 processos a comunicar (Pear-to-pear)

Comunicação direta assimétrica: Apenas quem envia diz para quem envia

Comunicação indireta (impessoal) : as mensagens são enviadas e recebidas por meio de "caixas de correio" ou "portos". Assim há uma maior variedade em como o canal de comunicação é estabelecido: Pode estar mais que um associado a um processo. Só é estabelecido se o par de comunicação possuir uma caixa de correio partilhada. Levanta um problema de 2 ou mais processos tentarem receber uma mensagem da mesma caixa de correio

Opções de envio e receção
Blocking send- o processo de send bloqueia até que a mensagem seja recevida pelo processo recetor e resume a operação
Blocking recieve- o processo bloqueia até uma mensagem ficar disponível.

Buffering: capacidade de armazenamento interno com o canal de comunicação em si.
Implementações do link de comunicação
Capacidade zero: Não há fila. Há blocking send.
Bounded capacity: a fila tem tamanho finito. Se a fila está cheia. O sender bloqueia até haver espaço.
Unbounded Capacity: a fila tem tamanho (potencialmente) infinito.

9Processor sheduling

Podemos ver a execução de um processo como uma sequencia alternada de momentos:
CPU burst- está a usar o CPU
I/0 burst- espera que um request I/O seja concluído

Classificação de processos:
I/o bound- tem vários CPU burst curtos
CPU bound- tem alguns CPU burst longos.
No multiprograming aproveita-se os I/O burst para por outros processos a utilizar o CPU.

Scheduling de longo prazo- determina que programas são admitidos para processamento. Controla o grau de multiprogramming. (NEW, TERMINATED)
Scheduling de curto prazo- decide qual processo vai ser executado a seguir(READY, Blocked, RUN)

- Escalonador preemptive: o processo pode perder o processador devido a razões externas. (tem o **time-out** e o **preempt**). Utilizar em sistemas reais e interativos.
- Escalonador não premptive: o processo mantem o processador até que este bloquei ou acabe. (**time-out** e o **preempt não existem**)

Critérios de escalonamento

- Critérios orientados ao utilizador - relacionados ao comportamento do sistema conforme percebido pelo utilizador ou processo individual

Turnaround time - intervalo de tempo entre a apresentação de um processo e sua

conclusão (inclui o tempo de execução real mais o tempo gasto esperando recursos, incluindo o processador). É a medida apropriada para um batch job. Deve ser minimizado.
Waiting time- soma dos períodos gastos por um processo que está a espera no estado Ready.
Response time- tempo desde o envio de um pedido até que resposta comece a ser recebida. Medida apropriada para um processo interativo.
Deadlines: tempo de conclusão de um processo.
Predictability- como a resposta é afetada pela carga no sistema. Um determinado trabalho deve ser executado em aproximadamente a mesma quantidade de tempo e com o mesmo custo independentemente da carga no sistema

- Critérios orientados para o sistema - relacionados com a utilização eficaz e eficiente do processador.

As **prioridades** podem ser:

Determinísticas: São definidas de forma determinística

- Estáticas: Comuns em Real Time Systems. Pode conduzir a starvation.São atribuídas prioridades fixas consoante a importância do processo.
- Mudança determinística: quunando um processo é criado, uma prioridade é atribuída ao processo. Se ocorrer um timeout a prioridade de processo decrementa. E se ocorrer um wait-event a prioridade é incrementada.

Dinâmicas: A prioridade do processo varia consoante o histórico de execuções passadas dos processos. São utilizados algoritmos de aging: atribuído um nível de prioridade em função do tempo em que o processo está à espera.

Políticas de scheduling:

Favorecer justiça: previne starvation
First Come First Served:
Não preemptive. Favorece CPU-bound. Pode resultar num mau uso de recursos.

Round robin: Preemptive. Favorece CPU-bounded. O seu desempenho depende do time quantum. Pode resultar num mau uso de I/O.

Shorteste process Next: não preemptive. Escolhe o processo com próximo CPU-burst mais curto.
Diminui o turnaround dos processos I/o intensivos.
Pode resultar em starvation para CPU-bounded. É necessário ultimar a duração do processo.

10 DeadLock

Condições para ocorrer deadlock(todas têm de acontecer):

- Exclusão mútua: apenas 1 processos pode utilizar um recurso de cada vez.
Para combater é necessário que os recursos sejam partilháveis.
- Hold and wait: um processo retém um recurso enquanto espera por outro que está a ser utilizado por outro processo.
Para combater é necessário que o processo peça todos os recursos necessários de uma vez. (pode ocorrer starvation). TB é possível se o processo largar os recursos na sua posse qnd não consegue aceder ao recurso que quer. (starvation e busy witing podem acontecer)
-No preemption- Apenas o processo com a posso do recurso o pode largar. Ninguém lho tira.

-Circular wait- um conjunto de processos de espera existem de tal forma que cada 1 está à espera de recursos que estão na posse de outros.
Para combater pode-se atribuir um id numérico diferente a cada recurso e impor que a aquisição do recurso tem de ser feita de ordem crescente ou decrescente (starvation pode acontecer)

Deadlock prevention

Se pelo menos 1 das condições em cima não se verificar, não há deadlock. É da responsabilidade dos processos e não do gestor de recursos não haver deadlock.

Nos filósofos, fazer com que cada garfo tenha o ID do filosofo à sua direita e depois cada filosofo busca o garfo com id mais pequeno previne circular wait.

Deadlock avoidance

É menos restrito que o prevention. Nenuma das condições é negada. O Gestor de recursos decide o que fazer em termos de alocação de recursos de forma a evitar deadlock. Requer conhecimento prévio.

Process Initiation denial: não começa o processo se o que ele pedir pode levar a deadlock. Um novo processo *q* só começa se:

$$C_q \leq R - \sum_{p \in P} C_p$$

Cp: recursos declarados pelo processo; R= quantidade de cada recurso

Resouce allocation denial: Não fornece o recurso a um processo se a sua alocação levar a deadlock. Um novo processo *q* só começa se o algoritmo de banqueiros se verificar:

$$C_{s(k)} - A_{s(k)} = V + \sum_{m=1}^{k-1} A_{s(m)}$$

Ap quantidade de recursos já alocados.

Deadlock detention

Quando não é feito prevention nem avoidance deadlok pode ocorrer. Pode se ignorar ou a cadeia circular tem de se romper:

- libertando recursos de um processo (guarda o estado do processo para mais a frente o voltar a chamar)
- rollback- se os estados da execução dos diferentes processos forem periodicamente guardados, um recurso é libertado de um processo cujo estado de execução é revertido para o momento em que o recurso lhe foi atribuído.
- matar processos

11 Memory management

Para ser executado, um processo tem de ter o seu espaço de endereçamento, pelo menos parcialmente, residente na memoria principal. O objetivo é controlar a tranferencia de dados entre a main memory e a secondary memory (swapping área).

Cache memory- pequena, rápida, volátil, dispendiosa (Cache dentro do processador). Vai ter uma copia das posições da memoria de intruções mais usadas pelo processador.

Main memory- media, volátil, boa relação velocidade de acesso-preço (RAM).

Memória secundária- grande, lenta,não volátil e barata (HDD/SSD)

- FileSystem- armazenamento para informação mais ou menos permanente.
- Swapping área- extenção da memoria principal de forma a que mais processos possam existir

Princípio da localidade de referência: Tendência de um programa a aceder ao mesmo set de locais da memoria repetidamente dentro de um curto espaço de tempo.

Stack overflow: Quando a área entre a região de dynamic variables e a stack, que normalmente não está alocada, se esgota no lado da stack resulta num erro fatal.

Espaço de endereçamento lógico: A imagem binária do espaço de endereçamento. É realocável.

Espaço de endereçamento físico: A região da memoria principal onde o processo é carregado para ser executado.

Existem 2 problemas que têm de ser resolvidos:

Mapeamento dinâmico - capacidade de converter um endereço lógico num endereço físico em tempo de execução, de modo que o espaço de endereço físico de um processo possa ser colocado em qualquer região da memória principal e **movido**, se necessário.

Proteção dinâmica - capacidade de impedir o acesso em tempo de execução a endereços localizados fora do espaço de endereço do próprio processo.

A) Memoria real

Numa organização de memoria real há um correspondência direta(um por um) entre o EE logico e o EE físico. Isto traz consequências:

- Limitação no EE do processo (não pode ser maior que a memoria)
- Contiguidade do EE físico;
- A swapping área é uma extenção da memoria principal (qnd há falta de espaço)

Existe uma peça de hardware que permite mapeamento e proteção dinâmicos- MMU contêm:

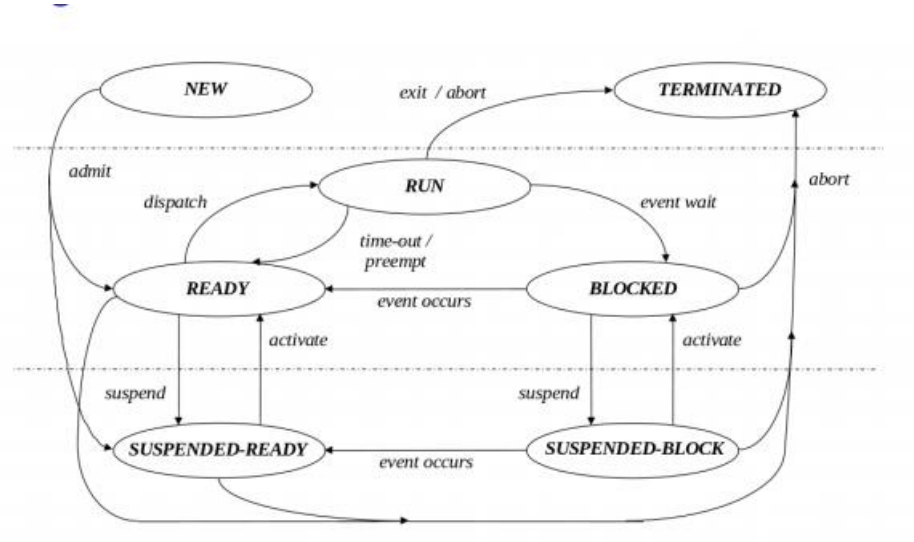
Limit register- que contêm o tamanho em bytes do EE logico

Base register- contêm o endereço inicial da região da memoria principal onde o EE físico do processo vai ser colocado.

Estes registos são carregados sempre que um processo passa ao estado run (dispatch).

Conversão de um endereço lógico num físico:

1) Endereço lógico é comparado com o limit register
   a) Se for menor então é uma referência valida endereço logico é somado ao base register;
   b) Se for maior ou igual é gerada uma exceção



Longo prazo

Admit: depois do processo ser criado, se houver espaço na memoria principal, o seu EE é lá carregado e o base register é atualizado com o enderenço inicial da região da memoria e o processo passa para ready.

Suspended: se não houver espaço, o seu EE é armazenado temporariamente na swapping área e o processo passa para suspended-ready área até o activate.

Médio prazo

Se for preciso mais memoria para outro processo, pode ocorrer swapped out de um processo blocked para libertar a memoria física que este está a utilizar. Base register fica indefinido. Se a memoria ficar disponível, um processo suspended pode ser swapped in.

Arquiteturas

Partições fixas: a memoria principal pode ser dividida num numero estático de partições, não necessariamente do mesmo tamanho. A partição de maior tamanha determina o tamanho do maior processo permitido.

- É fácil de implementar
- É eficiente: há pouca sobrecarga do SO
- O nº máximo de processos ativos é fixo.
- Há um unso ineficiente da memoria devido à fragmentação interna. A parte da partição não usada pelo processo é desperdiçada.

Há 2 politicas de escalonamento de processos possíveis:
Valorizar a justiça: o 1º processo na fila dos suspended ready cujo EE cabe na partição.
Valorizar a ocupação da memoria principal: o 1º processo da fila com maior espaço de endereçamento que caiba na partição. Para evitar starvation pode ser usado um mecanismo de aging.

Partições dinâmicas: Inicialmente é considerada toda a memoria disponível, constituindo um único bloco. Depois, vão se reservando partições de tamanho suficiente para o espaço de endereçamento de cada processo que chega. Quando já não é precisa, a partição é libertada.

- É geral- independente dos processos a executar.
- Baixa complexidade

- (mau) fragmentação externa e insuficiente: não é possível fazer algoritmos que são eficientes em alocar e libertar espaço ao mesmo tempo.

Como a memoria é reservada de forma dinâmica, o SO tem de manter um registo atualizado das áreas ocupadas e livres através de Listas biligadas:

Lista das regiões ocupadas- localiza as regiões reservadas para o alojamento do EE de processos residentes na memoria principal

Lista de regiões livres- localiza regiões ainda disponíveis.

Aqui já não faz sentido valorizar-se a ocupação da memoria logo a política é a valorizar a justiça.

Para além dos resíduos que podem sobrar das partições, aqui não há fragmentação interna. Mas é possível sofrer de um problema de fragmentação externa: (blocos livres não contíguos) podem ser criados buracos tão pequenos que não permitam o alojamento de novos processos.

Solução:

Recolha de lixo: compactar os espaços livres agrupando-os. É necessário parar todos os processos e reagrupá-los. Em memorias muito grandes é um processo demorado.

Escalonamento de espaços livres:

First Fit: a lista de espaços livres é percorrida desde o inicio até aparecer a primeira região com espaço suficiente para o processo.

Next fit: variante do first fit que começa no stop point da ultima pesquisa.

Best Fit: A lista é totalmente percorrida e é escolhido a menor região onde caiba o processo.

Worst fit: a lista é totalmente percorrida e é escolhida a maior região existente

B) Memória Virtual

O espaço de endereçamento logico e físico são totalmente dissociados, ou seja, não há limitações no espaço de endereçamento de um processo. Já podem existir metodologias que permitem executar processos com EE maior que a memoria principal. O espaço de endereçamento físico não é continuo de forma a garantir uma ocupação do espaço mais eficiente. Há uma extensão da memoria principal, a swapping área, de forma a manter uma imagem atualizada do EE dos processos que estão a coexistir, nomeadamente a sua parte variável.

Endereço lógico tem 2 campo: nblk-> identifica um bloco especifico e offset identifica a posição dentro do bloco, como um offtet desde o seu inicio.

MMU contêm:

Base register- representa o início do endereço da block table do processo.

Limit register- representa o número de entradas da tabela (nºde blocos)

Passos para acesso à memoria:

1) o campo nblk do endereço lógico é comparado com o registo limite da block table
   a) se for válido (≤), o registo base da block table mais nblk aponta para a entrada da block table, que é carregada na MMU
   b) se não (>), um acesso nulo à memória (ciclo fictício) é colocado em movimento e uma exceção é gerada devido ao erro de endereço
2) A flag M / AS é avaliada

a) se for M (o bloco referenciado está na memória), a operação pode prosseguir

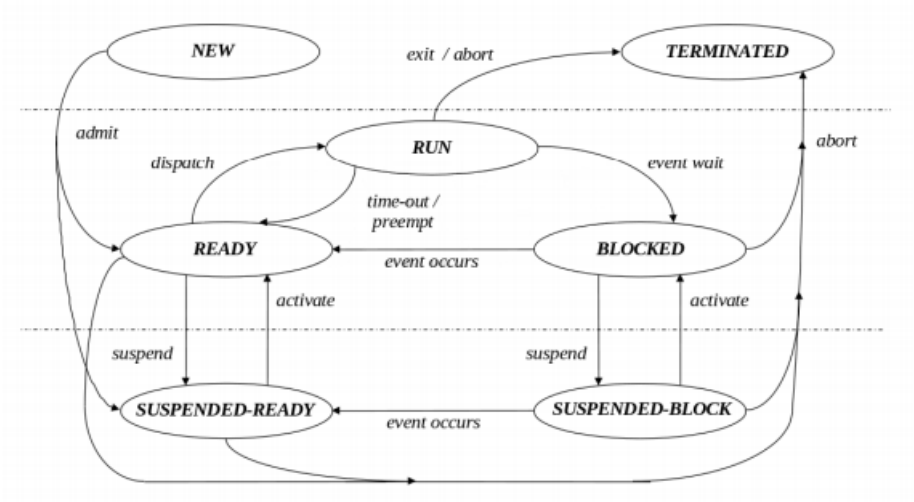
b) se não (o bloco referenciado é trocado), um acesso nulo à memória (ciclo fictício) é colocado em movimento e uma exceção é gerada devido à falha do bloco

3) O Campo offset do endereço lógico é comparado com o registo de limite de bloco

a) se for válido (≤), o registo de base do bloco mais offset aponta para o endereço físico

b) se não (>), um acesso nulo à memória (ciclo fictício) é colocado em movimento e uma exceção é gerada devido ao erro de endereço.

A memoria virtual introduz versatilidade, mas torna o acesso à memoria com um custo de 2 acessos. No entanto isto pode ser minimizado através do princípio da localidade de referência, assim a MMU tem o contudo das entradas da block table armazenado numa memoria interna chamada Translation Lookaside buffer (TLB-hardware). Assim o primeiro acesso pode ser um hit qnd a entrada é armazenada na TLB, e neste caso o acesso é interno ao processador. Ou um miss no caso da entrada não estar armazenada na TLB, havendo acesso à memoria principal.



Longo prazo

Qnd um processo é criado, o seu espaço de endereçamento é contruído e pelo menos a sua parte variável é posta na swapping área e a sua block table é organizada. Alguns blocos podem ser partilhados com outros processos.

Admit: depois do processo ser criado, se houver espaço na memoria principal, pelo menos a sua block table, o primeiro bloco de código e o block da sua stack são carregados. As entradas da block table são atualizadas e o processo passa para ready. Se não processo passa para suspended-ready- suspend.

Curto prazo:

Event_wait: Durante a execução do processo(RUN), como vão haver blocos que se pretendem aceder indisponíveis, vai acontecer block fault e o processo é colocado em blocked enquanto o bloco não é swapped in. Event_occurs: Quando o bloco está em memória o processo passa a ready,

Médio prazo

No estado ready ou blocked, todos os blocos de um processo podem ser swapped out, se for preciso memoria.

Activate: Se a memoria ficar disponível, um processo suspended/bloked pode ser swapped in e passar para ready/blocked. As correspondentes entradas da block table são atualizadas.

C) Paginação

A memória é dividida em pedaços iguais de tamanho fixo, chamados frames (potencia de 2):
O espaço de endereço de um processo é dividido em blocos de tamanho fixo, do mesmo tamanho, chamados de páginas. O tamanho das frames é igual ao tamnho das paginas. Esta divisão não é totalmente cega, o linker geralmente inicia uma nova página quando uma nova região começa.
Num endereço lógico:
• os bits mais significativos representam o número da página
• os bits menos significativos representam um offset dentro da página
(2base registers)
A tabela de paginação pode representar a totalidade do espaço de endereçamento de um processo. E assim deixa de haver necessidade de saber quantas páginas têm. Assim, não há limit register associado ao tamanho da tabela de páginas. Isto pode maximizar o vazio que existia entre a stack e a heap memory. Tambem não é necessario validar o offset pq este já tem um valor predefinido.

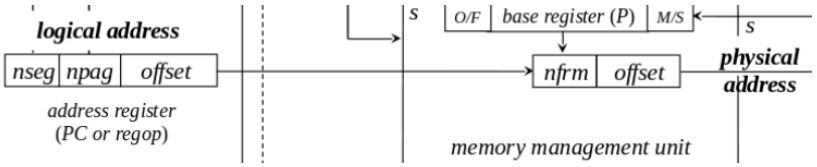
- Geral
- Boa utilização da memoria- não conduz a fragmentação externa e a interna é negligenciável (Não deixo lá buracos).
- Não tem requisitos especiais de hardware
- Acessos à memoria mais demorados, por causa de um acesso à priori à page table (TLB tb minimiza o impacto)
- Operabilidade muito exigente.

D) Segmentação

Normalmente o EE de um processo tem diferentes segmentos: code, static variables; heap memory, shared memory, stack…
Numa arquitetura de segmentação, os segmentos de um processo são manipulados separadamente. Pode ser usada partição dinâmica para alocar cada segmento, e como consequência um processo pode não se contiguo na memória e alguns segmentos podem até nem estar na memoria principal.
Sozinha, a segmentação pode resultar em fragmentação externa e um segmento crescente pode impor uma mudança na sua localização.

E) Combinação de segmentação com paginação

Pode resolver problemas de cima, mas introduz complexidade:
1º O EE logico de um processo é partido em segmentos
2º Cada segmento é dividido em páginas;
A MMU deve conter 3 base registers e 1 limit register. 1 base e 1 limit para a segmentation table; 1 base para a page table ; 1 base para o memory frame.



Passos para acesso à memoria:

1) Acesso à segmentation table
2) Acesso à page table
3) Access ao endereço físico

Substituição de páginas

Numa arquitetura de paginação (ou combinação de segmentação e paginação), a memória é partida em frames, cada um com o mesmo tamanho de uma página. Um frame pode estar livre ou ocupado (contendo uma página)
Uma página na memória pode estar:
• bloqueada - se não puder ser removida da memória (kernel, cache de buffer, arquivo mapeado na memória)
• desbloqueada - se puder ser removido da memória
• Se não houver frame livre disponível, um ocupado pode precisar de ser libertado
Este é o propósito da substituição de página
• A substituição de página só se aplica a páginas desbloqueadas e era ótima se se podesse selecionar a página onde o tempo para a próxima referência era mais longo.

Políticas de substituição:
Least Recently Used: seleciona o frame que não foi referenciado há mais tempo. É preciso percorrer a lista de frames ocupados toda (+hardware, pouco eficiente, alto custo de implementação)
Not recently used: Seleciona um frame baseando-se em classes definidas pelos bits ref e mod(campo da entrada da page table). Seleciona um frame da classe não cheia mais baixa. O sistema atravessa periodicamente a lista de frames ocupados e mete o ref a zero.
FIFO: Seleciona baseando-se no tamanho da memoria “stay in”. O frame com a pagina mais antiga é selecionado. (A assunção de que a lista de frames ocupados é um fifo é falível)

Segunda chance (melhoramento do fifo): o frame com a pagina mais antiga é selecionado. Se o bit ref é 0, a seleção é feita. Se não o bit é resetado, o frame é colocado outra vez no fifo e o processo repete-se no próximo frame. (nem sempre um ref=0 é encontrado).

Clock: A lista é transformada em circular e há um ponteiro que indica o elemento mais antigo. Quando REF != 0 é resetado e o ponteiro avança. Quando REF=0 o frame é selecionado e o ponteiro fica na próxima fase.

Working set: 2 páginas iniciais do processo: 1 com a 1ª instrução e 1 com o inicio da stack;

Quando um processo passa a ready qual pagina deve ser alocada na memoria principal?

- Demand paging- nenhuma, espera-se pelo page fault;(ineficiente)
- Prepaging- as que são mais prováveis de serem referenciadas. Numa primeira vez deve ser o working set. Depois enviar aquelas que estavam na memoria principal quando o processo foi suspenso.