1. **Programa:** Descrever o que é um programa, sua natureza e função.

Podemos começar trazendo as definições de software e programa, esses conceitos são muito relacionados, porém não são a mesma coisa. Programa como ensina Siberschatz “é uma entidade passivo, como um conteúdo de um arquivo armazenado em disco” (pag. 64), ou ainda como cita Maziero “software de aplicação é representado por programas destinados ao usuário do sistema, que constituem a razão final de seu uso” (pag. 2), ele ainda nos apresenta exemplos: programas editores de texto, jogos e navegadores de internet, eu ainda ouso a citar a interface de usuário do Linux, o KDE, me referenciando as aulas do professor Fausto.

Francis B. Machado ao tratar de threads define que “um programa é uma sequência de instruções, compostas por desvios, repetições e chamadas a procedimentos e funções” (pag. 82), na internet é muito comum acharmos a definição de que programa é um conjunto de instruções concatenadas e escritas em uma determinada linguagem que tem alguma função especifica e que será executada em um computador. Ou seja, é um objeto estático quando inativo porem ao ser executado dará origem a algo dinâmico que podemos chamar de processo, isto é, ele é algo estático e inativo, porém ao ser executado deverá instruir o computador a executar operações para qual ele foi criado.

Já software é algo mais abrangente, e não inclui somente programas, inclui também sistemas operacionais, dados, drivers, bibliotecas, ferramentas de desenvolvimento entre muitos outros. Pode ser dividido em software de sistemas, de aplicativos e software de programação. Como exemplo se software podemos citar os sistemas operacionais que atuam na camada modo núcleo, entre o hardware e os aplicativos e que tem a função de gerenciar o uso dos componentes de hardware, fornece uma camada de abstração para os programas e fornecer uma interface de acesso para dispositivos com tecnologias distintas como usb e ide.

1. **Processo:** Apresentar a definição de processo, seus componentes principais e características.

Citando Voltz, “um processo pode ser considerado um programa em execução”, quando um programa está estático, ou seja, ele ainda não foi iniciado, esse programa não gerou nenhum processo pois ainda não precisou de recursos. Assim que um programa é executado recursos são alocados para ele e este dá início a um ou vários processos.

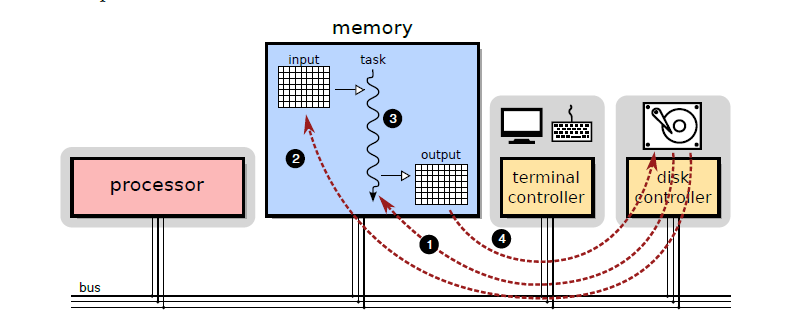


Fig. 01 – Execução de uma tarefa em um Sistema Monotarefa.

Computadores em lote somente executam um processo por vez e somente após sua finalização outro processo será executado. Isto não ocorre com computadores pessoais, seria impossível assistir um vídeo e ouvir o seu áudio ao mesmo tempo se somente um processo fosse executado por vez.

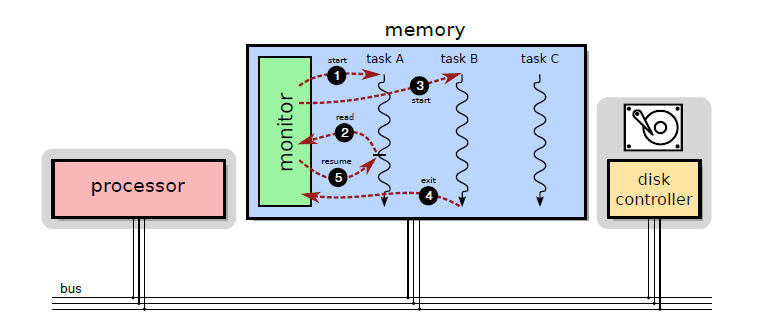


Fig. 02 – Execução de uma tarefa em um Sistema Multitarefas

Algumas características são inerentes ao processo, como já citado o processo é uma abstração do programa que ao ser executado inicia-se sua função determinada no código fonte do programa, se um programa foi executado 2 vezes cada execução gera um processo diferente. Os processos podem ser executados em primeiro plano, como no caso de um editor de texto, ou em segundo plano como no caso do spool de impressão que fica aguardando ser chamado para executar suas tarefas determinadas.

Uma diferenciação pode ser identificada levando em conta as características dos processos, desta forma, dividindo o processo em contexto de software, hardware e memória podemos identificar algumas características peculiares para o melhor estudo do caso.

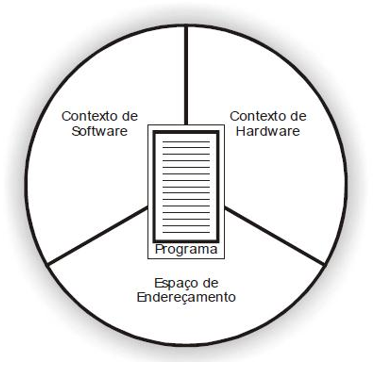


Figura 03: Características da estrutura de um processo

Sob o contexto de software quando o processo é criado é necessário identifica-los com algumas características próprias, como nome, processo ID, usuário ID, prioridade de execução, data e hora da sua criação, tempo de cpu, quota e privilégios por exemplo.

Sob o contexto de hardware esse processo é identificado no registrador de propósito geral, no program conter, no ponteiro de pilha e registrador de status. Todas essas informações de programa serão alocadas em uma estrutura de dados chamada de pcb (process control block). Sempre que um processo é criado, é criado também esse pcb do processo aonde o sistema operacional estará acessando essa estrutura para acessar todas as informações vinculadas ao processo.

Informações como gerenciamento de memória (registradores base e limite, tabelas de páginas ou de segmentos, informações de status de E/S (lista de arquivos abertos, lista de dispositivos alocados ao processo e etc.), informações de contabilização (tempo de execução real e de cpu) e informações de escalonamento (prioridades, ponteiros para filas de escalonamento, parâmetros).

Espaço de endereçamento é o espaço que o programa vai usar para ser alocado na memória principal. E no contexto de memória todo processo que é executado recebe endereços de memória aonde serão alocados. Quanto alocado na memória principal esse processo está dividido em 4 grandes áreas:

1. Pilha que contém os dados temporários, é utilizada para gerenciamento de funções e procedimentos, assim como os argumentos de entrada das funções como também os argumentos de saída ou retorno.
2. Heap que representa o espaço para alocação dinâmica de memória durante a execução do processo, ou seja, dados manipulados por ponteiros na linguagem c, por exemplo, ou objetos em outros linguagens de programação.
3. Região de Texto ou Instruções é a região de código, aonde vão estar armazenadas todas as instruções de código do programa, ou seja, todas as instruções binárias que serão executadas pelo processador. A busca de instruções pelo processador será feita nessa região da memória e carregada no registrador interno do processador.
4. Região de dados ou global é a região aonde será alocada todas as variáveis globais e todas as constantes declarado no programa.

A região de pilha e de heap vão compartilhar a mesma região de memória, a pilha começará com os endereços mais altos e a heap com os endereços mais baixos. Pilha decrementa os endereços e a pilha vão em ordem crescente.

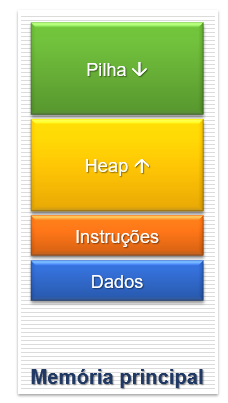


Fig. 04: Processo alocado em memória

Podemos citar quatro formas em que os processos podem ser criados. O primeiro caso é quando ligamos o computador e o sistema operacional é iniciado, desta forma alguns programas são automaticamente iniciados e conseguintemente os processos relativos a eles.

Outra forma é pela requisição de um usuário, por exemplo, quando o usuário deseja editar um arquivo e executa o mesmo, o sistema operacional irá buscar o programa padrão para editor desse tipo de arquivo, ao encontrar o iniciará, desta forma iniciando os processos atrelados ao programa.

Seguindo nesse mesmo contexto pode-se criar um processo por um outro processo em execução, ao terminar de digitar seu texto o usuário deseja imprimi-lo, desta forma solicita ao editor de texto que envie este documento para a impressora, neste momento é criado um outro processo pela inicialização do gerenciador de impressão.

A quarta forma é no caso de tarefas em lote (batch job), somente utilizadas em sistemas de grande porte, geralmente esses sistemas executam uma tarefa por vez, assim ao término de uma tarefa o sistema operacional iniciará o próximo processo.

Um processo não pode ficar eternamente em execução e o seu término poderá ocorrer também em 4 situações distintas.

O primeiro caso de término é quando o processo decorre normalmente, executa todas as suas funções com êxito ou quando o usuário solicita seu encerramento voluntariamente, a este termino de processo podemos chama-lo de término normal voluntário.

A segunda forma de finalização de um processo, é chamada de término por erro voluntario, ou seja, é uma finalização que ocorre por um erro esperado, Voltz cita o exemplo de quando o usuário tenta abrir um arquivo que já não existe mais fisicamente, assim o processo informa que o arquivo não existe mais e finaliza a chamada de abertura de arquivo.

Já uma outra forma de finalização é o de erro fatal involuntário, este ocorre quando ocorre um erro não tratado na execução de um processo que na qual executou uma instrução ilegal ou não planejada. Podemos citar uma calculadora que fez uma divisão por zero não tratada ou programas que travam do nada e que causam pânico ao usuário desavisado.

A quarta e última situação de término de um processo é quando um outro processo pede seu cancelamento e aquele se encerra de forma involuntária, como por exemplo pelo comando kill do Linux que encerra um processo pelo seu PID (Identificador do processo), ou no Windows quando no gerenciador de tarefas pedimos como o comando finalizar tarefa que um processo se encerre.

Ainda sobre processos precisamos mencionar os cinco estados em que o processo pode ser encontrar ou seja o ciclo de vida do processo: Novo, em execução, em espera, pronto e concluído.

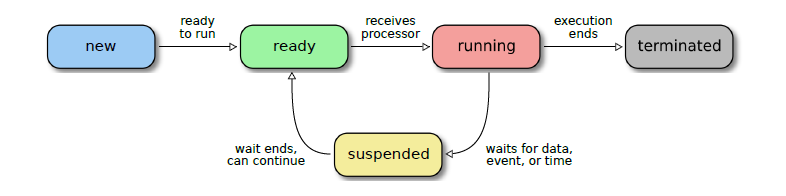


Fig. 05 – Diagrama de estados de uma tarefa, sistema multitarefas.

Novo, ou seja, o processo está sendo criado, seu código foi carregado na memória junto com toda sua estrutura para devida execução

Pronto, o processo já foi carregado em memória, está aguardando para ser executado ou retornar a ser executado, só esperando ser chamado pelo escalonador que o colocou numa fila de espera de processos.

Executando, ou em execução, o processo nesse momento está sendo executado pelo processador, ou seja, suas instruções estão sendo efetuadas.

Suspensa, devido a necessidade de aguarda dados externos o processo foi colocado em estado de suspensão pelo escalonador até que a sincronização dos dados seja terminada ou que o tempo de suspensão seja encerrado.

Terminada, ou concluído, o processo da tarefa terminou e agora ela já pode ser descarregada da memória.

Tão importante quanto os estados da tarefa são as transições de estado

...->Nova, quando uma nova tarefa começa a ser preparada para executar.

Nova->Pronta, quando o processo já foi carregado todo na memória.

Pronta->Executando, quando o processo foi selecionado pelo escalonador para ser executado.

Executando->Pronta, quanto o tempo de uso do processador foi exaurido e ele não precisa aguardar nenhum outro processo, assim ele retorna ao fim da fila para aguarda a continuação da execução, ou acontece um outro tipo de interrupção.

Executando->Suspenso, quando o processo precisa de um outro recurso que não está disponível no momento, uma sincronização ou um dado externo, o escalonador o coloca em suspensão momentânea.

Suspensa->Pronta, quando o recurso se torna disponível e o processo já pode voltar para o estado de pronto, retornando a fila para aguardar ser processado.

Executando->Terminada, quando a tarefa é encerrada pelo seu fim ou por algum erro.

Terminada->..., quando o processo já foi terminado e pode ser descarregado da memória.

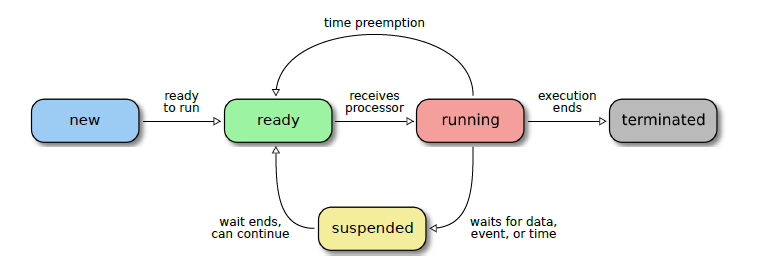


Fig. 06 – Diagrama de estados de uma tarefa, sistema

multitarefas. Preemptivo por tempo

A matéria sobre processos é fascinante e extensa, podemos citar mais temas de estudo como: tipos de escalonadores de processos, processos que rodam background (deamons) (segundo plano) e foreground (primeiro plano) e árvores de processos.

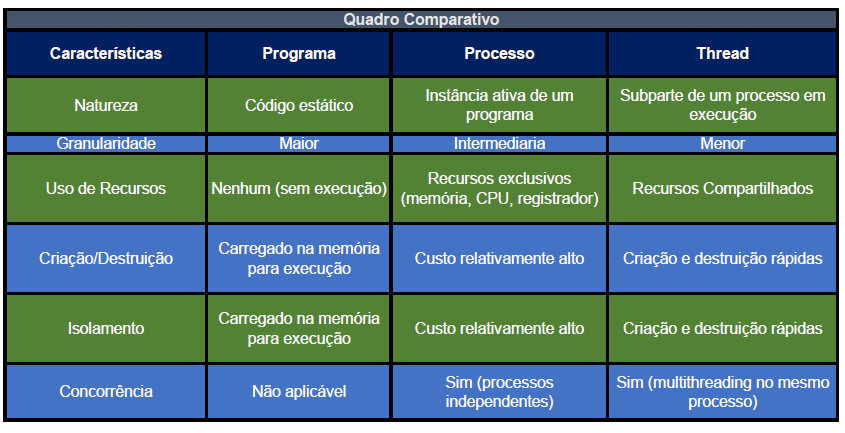


Fig.07 – Quadro comparativo

1. **Thread:** Conceituar thread, destacando suas características, diferenças em relação a processos e relevância na programação.

Antigamente, antes do final da década de 1970, os computadores suportavam apenas processos com um único thread, ou seja, um programa era executado iniciando um único processo que fazia parte do seu contexto.

Foi durante o desenvolvimento de um sistema operacional chamado de Toth, 1979, que foi apresentado um conceito novo de processo, lightweight ou processo leve, aonde pela primeira vez existia uma área na memória que poderia ser compartilhada entre processos. Porém somente mais tarde, em 1980, com o desenvolvimento do sistema operacional Mach, da Universidade de Carnegie Mellon, que o conceito de separação entre processo e thread ficou mais clara.

Maziero define thread “como sendo um fluxo de execução independente. Um processo pode conter uma ou mais threads, cada uma executando seu próprio código e compartilhando recursos com os demais threads localizados no mesmo processo” (pág. 58).

Maziero complementa e nos informa com é uma estrutura de thread, ele nos ensina que “cada thread é caracterizado por um código em execução e um pequeno contexto local, o chamado Thread Local Storage (TLS), composto pelos registradores do processador e uma área de pilha de memória, para que a thread possa armazenar variáveis locais e efetuar chamadas de funções” (pag. 58).

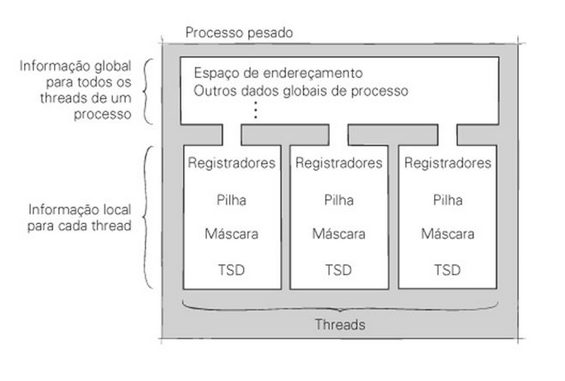


Fig. 08 - Relação entre processo e threads

Para Machado foi “a partir do conceito de múltiplos threads (multithread) é possível projetar e implementar aplicações concorrentes de forma eficiente, pois um processo pode ter partes diferentes do seu código sendo executadas concorrentemente, com um menor overhead do que utilizando múltiplos processos.” (pag. 82), desta forma a comunicação entre as linhas de execução ficam mais rápidas aumentando o desempenho da aplicação.

Atualmente o conceito de multithreads e seus benefícios são explorados por diversos sistemas operacionais, como menciona Silberchtz “muitos sistemas operacionais modernos agora oferecem recursos para que um processo contenha múltiplos fluxos de controle, ou threads” (pag. 82).

Em um ambiente monothread, um processo contém somente um programa no seu espaço de endereçamento, nesse caso se for necessário existir aplicações concorrente outros processos deverão ser implementados ou subprocessos. O problema desse tipo de implementação é que o uso de processos concorrente demanda a utilização de muitos recursos alocados, isto quer dizer que o consumo e tempo gasto do processador é muito elevado.

Como não existe compartilhamento de espaço de memória ou de recursos entre processos, a comunicação entre eles é difícil e lenta, pois cada processo precisa criar sinais, semáforos, memoria compartilhada ou troca de mensagem para haver a comunicação, e isso para ser eficaz, não ocorre de forma muito simples, pois cada processo possui seu próprio contexto de software, contexto de hardware e espaço de memória.

Já em um ambiente com múltiplos threads, o processo tem pelo menos um thread de execução, mas pode compartilhar o seu espaço de endereçamento com várias outras threads.

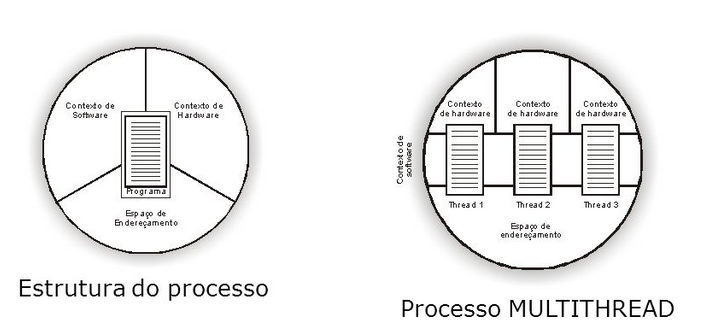


Fig. 09 – Comparação entre Processo e ambiente Multithreads

Machado nos ensina que “de forma simplificada, um thread pode ser definido como uma sub-rotina de um programa que pode ser executada de forma assíncrona, ou seja, executada concorrentemente ao programa chamador” (pág. 84). Basta o programador especificar e associar as sub-rotinas assíncronas com uma determinada thread, desta forma existirá threads concorrentes de sub-rotinas dentro de um mesmo processo, rodando desta forma múltiplas rotinas de uma forma leve.

As grandes vantagens de se utilizar um ambiente multithreads é a possibilidade de minimizar a alocação de recursos do sistema, diminuir o overhead na criação, troca e eliminação de processos.

Tanto os threads como os processos passam por trocas de contexto. Cada thread possui sua área de contexto de hardware individual, armazenada nos registradores do processador.

As Threads são implementadas internamente por meio de uma estrutura de dados chamada de TCB (Thread Control Block), que armazena além do seu contexto de hardware algumas outras informações como prioridade, estado de execução e bits de estado.

Vale muito salientar que nesse ambiente, a unidade de alocação de recursos é o processo onde todos os threads compartilham o espaço de endereçamento, descritores de arquivo e dispositivos de E/S. Porém cada thread representa uma entidade de escalonamento separada sendo que desta forma o sistema pode selecionar um thread independente para cada execução de uma sub-rotina.

A grande diferença entre processos independentes, subprocessos e threads é que os dois primeiros possuem espaços de endereçamento individuais e protegidos enquanto os threads compartilham essas áreas dentro de um mesmo processo, tornando a troca de dados entre os threads mais simples e rápida.

Ensina Machado que “para obter os benefícios do uso de threads, uma aplicação deve permitir que partes diferentes do seu código sejam executados concorrentemente de forma independente. Se um aplicativo realiza várias operações de E/S e trata eventos assíncronos, a programação multithreads aumenta seu desempenho até mesmo em ambientes com um único processador” (pag. 89).

Desta forma são diversos os benefícios de se utilizar um ambiente multithreads como já citamos, porém devemos citar alguns casos concretos para enfatizar bem essas vantagens.

A utilização de tarefas em background enquanto aplicações esperam operações de E/S para serem processadas, ficam mais ágeis com multithreads, assim como a utilização de processador, discos e outros periféricos de forma concorrente. Em ambientes clientes-servidor, threads podem solicitar o serviço remoto, enquanto aplicações podem ficar realizando outras tarefas simultaneamente. SGBDs, servidores de arquivos de impressão, servidores de correio e web são outros exemplos de aplicações que o uso de multithreads irá proporcionar grandes benefícios e vantagens.

Existem diversas abordagens na implementação de threads e que irão influenciar no desempenho, na concorrência e na modularidade de uma aplicação.

Os threads podem ser usados para implementar fluxos de execução dentro do núcleo do sistema operacional (threads de núcleo), podem ser oferecidos por uma biblioteca de rotinas fora do núcleo do sistema operacional (modo usuário), uma combinação de ambos (modo hibrido) ou por um modelo conhecido por scheduler activations.

No modo usuário, o sistema operacional não sabe da existência de múltiplos threads, sendo de responsabilidade exclusiva da aplicação gerenciar e sincronizar os diversos threads existentes. Para isso deve haver uma biblioteca de rotinas que possibilitem a aplicação criar e eliminar threads, por exemplo, como também troca de mensagens entre threads e os seus escalonamentos.

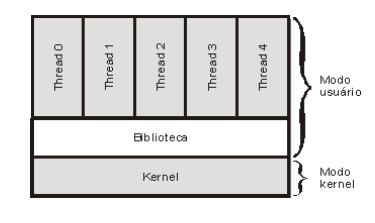


Fig. 10 – Modo Usuário.

Uma vantagem do modo usuário e poder criar threads mesmo que o sistema operacional não de suporte.

Threads em modo kernel já são bem diferentes, como essa threads são implementadas diretamente pelo núcleo do sistema operacional este já fornece todo um suporte para gerenciamento e sincronização. O sistema operacional conhece os threads criadas e cuida individualmente de seus escalamentos, desta forma se houver múltiplos processadores e em um processo tiver diversos threads o sistema operacional os renderizam para que haja a execução simultânea de vários threads ao mesmo tempo.

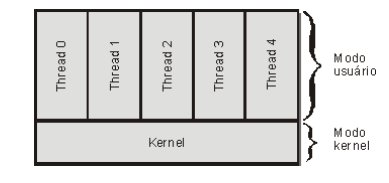


Fig. 11 – Modo Kernel.

Um problema do sistema de tratamento de threads a nível de kernel é que cada tratamento necessitará de um recurso do sistema operacional diminuindo o seu desempenho em relação ao tratamento em modo usuário somente.

Pensando nisso, foi criado um tratamento de threads em modo hibrido, que combina as vantagens dos threads implementados em modo usuário e modo kernel. Desta forma o núcleo do sistema operacional irá reconhecer os threads criadas em modo usuário e os poderá gerenciar e sincroniza-los, ou seja, escalonando individualmente.

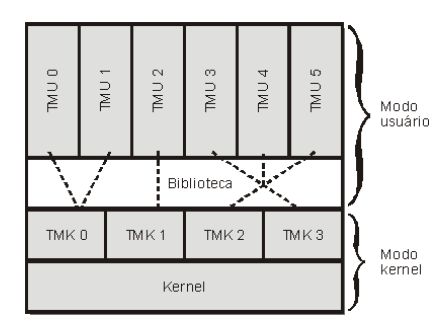


Fig. 12 – Modelo Hibrido.

Podemos enfatizar que assim como os threads herdam os benefícios dos seus modos, também herdam alguns de seus problemas, como o bloqueio de os threads de usuário em determinadas instancias e também os problemas dos desempenhos dos threads em modo kernel caso forem utilizar muitas ao mesmo tempo.

Pesquisando sobre o problema desse modo hibrido, em 1990 na universidade de Washington foi introduzido um modelo que combina o melhor das duas arquiteturas, a solução foi criar um sistema de troca de informações entre o kernel do sistema operacional e as bibliotecas de threads utilizando uma estrutura de dados chamada scheduler activations.

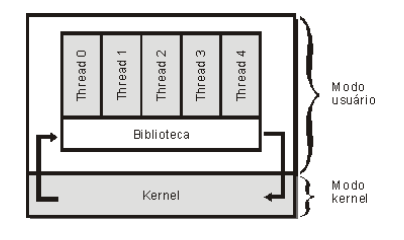


Fig. 13 – Modelo Scheduler Activations

Como a biblioteca em modo usuário e o kernel se comunicam e trabalham de forma cooperativa, se um thread utilizar uma chamada ao sistema que o colocará em modo de espera, não é necessário que o kernel seja ativado, desta forma não bloqueando os demais recursos de usuário, pois essa chamada será tratada diretamente pela biblioteca do processo que irá escalonar diretamente uma outra thread.

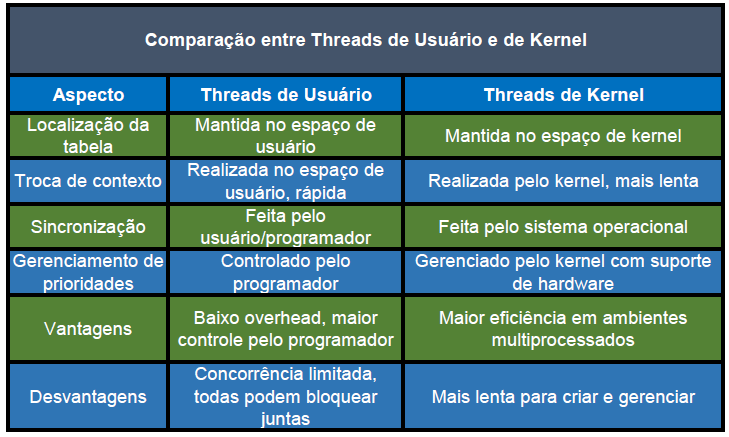


Fig. 14 – Quadro comparativos

Referências:

Machado, Francis Berenger. – Arquitetura de Sistemas Operacionais – 5ª ed. -Rio de Janeiro: LTC, 2013.

Maziero, Carlos Alberto. - Sistemas operacionais: conceitos e mecanismos – Curitiba: DINF - UFPR, 2019.

Sílberschatz, Abraham, Galvin, Peter, Gagne, Greg. – Fundamentos de Sistemas Operacionais – 9ª ed. – Rio de Janeiro: LTC, 2015.

Voltz, Wagner Mendes. – Sistemas Operacionais – Maringá: Unicesumar, 2018.

<<https://geekconectado.com.br/software-o-que-e-tipos-e-exemplos/>> acesso 12/09/2024 – autor: Dayse Branco

<<https://www.youtube.com/watch?v=jFgTdn5Qu4w&list=PLBw9d_OueVJTEpM8sOKJ7S8jleQPCSQs3&index=21&ab_channel=Prof.Santiago-Programa%C3%A7%C3%A3oeCi%C3%AAncia>> acesso 12/09/2024 – autor: Prof. Santiago - Programação e Ciência