Matéria:

* Revisão da matéria lecionada

**1 – Algoritmo do banqueiro *e míngua***

Recorde o que foi lecionado relativamente ao *algoritmo do banqueiro*, em que é aplicado para um sistema computacional gerir como os processos utilizam os recursos computacionais disponíveis (tais como tempo de CPU, memória). Este algoritmo serve, no contexto dos sistemas operativos, para realizar a alocação de recursos e evitar a ocorrência de bloqueios/impasses (*deadlock*) ao simular a alocação de uma quantidade máxima e pré-determinada de recursos que permitirá verificar os estados possíveis e testar possíveis atividades.

Considerando a seguinte implementação parcial do algoritmo em Java, complemente-o.

for (int i = 0; i < n; i++) {

// first find a thread that can finish for (int j = 0; j < n; j++) {

if (!finish[j]) {

boolean temp = true;

for (int k = 0; k < m; k++) { if (need[j][k] > work[k])

temp = false;

}

if (temp) { // if this thread can finish finish[j] = true;

for (int x = 0; x < m; x++) work[x] += work[j][x];

}

}

}

}

Uma imagem com texto, captura de ecrã, Tipo de letra, documento

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra, ecrã

Descrição gerada automaticamente

Adicionalmente, responda às seguintes questões:

1. Considere um sistema computacional que executa 2000 tarefas por mês e não possui qualquer esquema para evitar ou prevenir impasses/bloqueios. Os *deadlocks* ocorrem cerca de duas vezes por mês e o operador deve terminar e voltar a executar cerca de dez tarefas por *deadlock*. Cada tarefa vale cerca de dois euros (em tempo de CPU), e os trabalhos terminados tendem a estar a meio da execução quando são abortados. Um programador de sistemas estimou que um algoritmo para evitar impasses (como oalgoritmo do banqueiro) poderia ser instalado no sistema com um aumento de cerca de 10% no tempo médio de execução por tarefa. Como a máquina tem atualmente 30% de tempo de inatividade, todas os 2000 tarefas por mês ainda podiam ser executadas, embora o tempo de resposta aumentasse cerca de 20%, em média. Indique quais as os argumentos que suportam a incorporação do algoritmo e os argumentos de não o incorporem.
2. Podemos incorporar o algoritmo se:
   * A estabilidade e a confiabilidade do sistema são prioridades para a operação.
   * Há planos de uso futuro mais intensivo da máquina, onde o aumento da eficiência evita problemas de deadlock em uma escala maior.
   * O impacto financeiro dos deadlocks (em termos de interrupção de operações) é considerado mais relevante que o impacto em desempenho.
3. Não podemos incorporar o algoritmo se:
   * O impacto econômico e operacional dos deadlocks é desprezível (40 euros mensais e apenas 0,1% das tarefas são afetadas).
   * O aumento no tempo de resposta (20%) é problemático, e a simplicidade atual do sistema é uma vantagem que compensa a raridade de deadlocks
4. Um sistema consegue detetar que algumas dos suas *thread* estão no estado de míngua (*stravation*)? Se responder “sim”, explique como pode acontecer. Se responder “não”, explique como é que o sistema pode lidar com o problema da míngua (*starvation*).

Sim, um sistema pode detetar que algumas das suas threads estão no estado de míngua ou seja (starvation).

O sistema pode detetar míngua (starvation) através da monitorização do tempo de espera das threads para aceder a um recurso partilhado. Se uma thread permanecer bloqueada durante um período significativamente longo em comparação com as outras, isso pode ser um indício de (starvation).

Uma das formas de deteção passa por manter um registo do tempo que cada thread passa na fila de espera. Caso esse tempo ultrapasse um determinado limite, pode ser sinal de que a thread está a sofrer de (starvation). Outra abordagem envolve a monitorização do número de execuções: se algumas threads são constantemente agendadas enquanto outras raramente o são, pode indicar um problema de prioridade injusta.

Além disso, mecanismos de tracing e logging permitem registar padrões de execução das threads, possibilitando a identificação de casos de (starvation) através da análise de logs.