EP03 – SIMULADOR DE MEMÓRIA

MAC0422 - Sistemas Operacionais

Professor: Daniel Batista

Aluno: Francisco Membrive

Variáveis do programa

- Usadas por todos os algoritmos:
  - int tam - tamanho atual do bloco livre de memória lido
  - int inicio - índice atual correspondente ao início do bloco
- First Fit:
  - Sem necessidade de variáveis adicionais
- Next Fit:
  - int last_position: índice da posição que o next fit deve iniciar a busca
- Best Fit:
  - int best_tam: tamanho do menor bloco maior que m encontrado
  - int best_inicio: índice onde começa o bloco de tamanho best_tam
- Worst Fit;
  - int worst_tam: tamanho do maior bloco maior que m encontrado
  - int worst_inicio: índice onde começa o bloco de tamanho worst_tam

Experimentos

Os experimentos foram realizados em uma máquina Windows 10 Pro, usando WSL com Ubuntu 22.04.5 LTS. A máquina conta com um Ryzen 5 3600 de 6 núcleos e com 32GB de memória RAM DDR4, com 16GB reservados para o WSL durante a execução. Todos os experimentos foram feitos utilizando a configuração inicial de ep3-exemplo01.pgm, de modo que todos os resultados expostos aqui têm esse viés.

Os testes usando os traces entregues computaram o tempo de execução em milissegundos, com 30 execuções por combinação de parâmetros e intervalo de confiança de 95%. Além disso, foi feito um gráfico com o número de falhas de alocação de cada algoritmo em cada trace, sem intervalo de confiança pois o código é determinístico.

Além disso, foi realizado um experimento com 500 traces que faziam 600 requisições, gerados aleatoriamente, que computou o número de falhas de cada algoritmo e registrou o que desempenhou melhor (menos falhas) em cada trace.

Construção dos traces

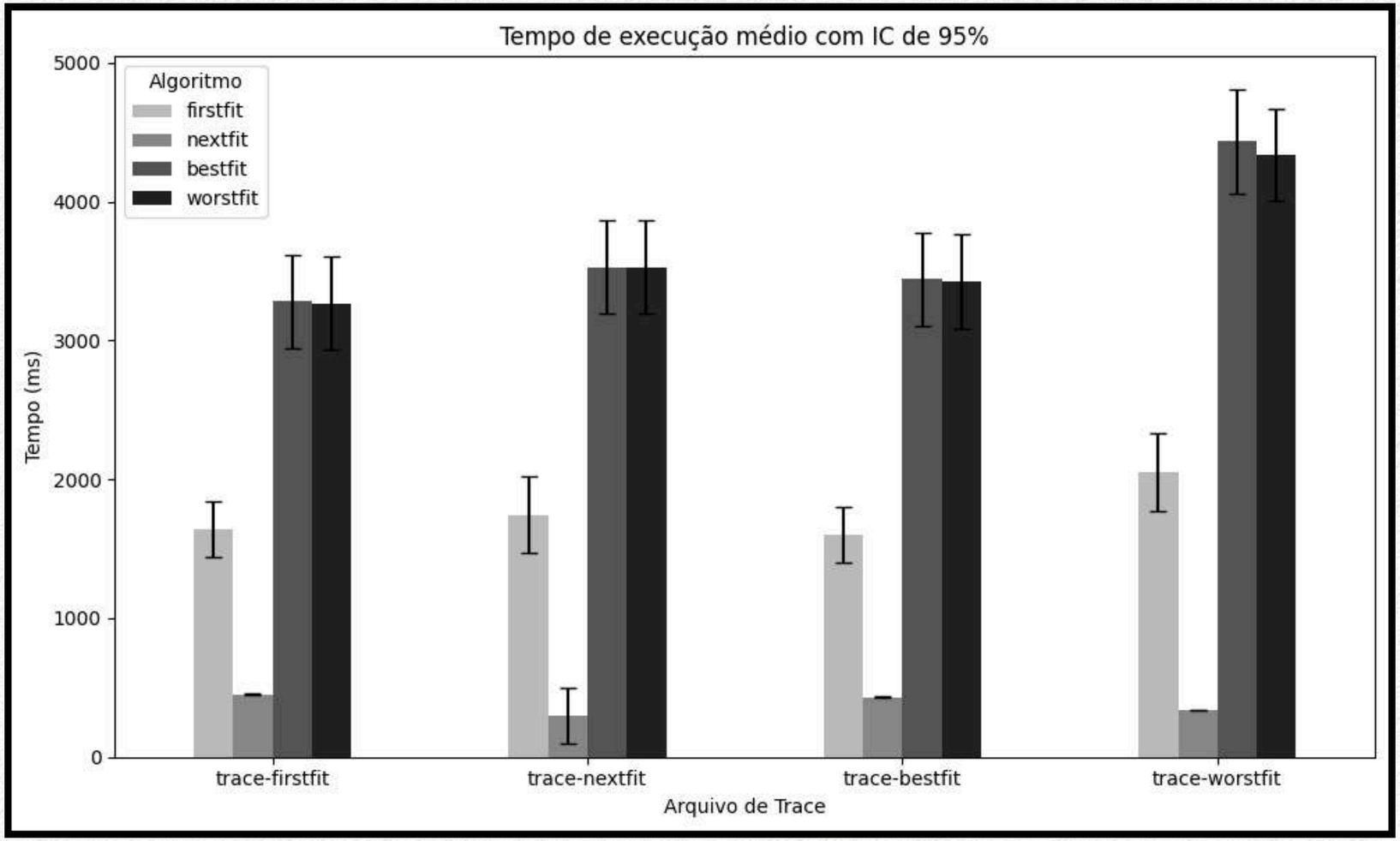
No geral, todos os traces foram construídos a partir de um script que mapeou as posições livres no arquivo ep3-exemplo01.pgm.

A partir disso, os blocos livres foram seccionados em requisições de tamanho máximo 256.

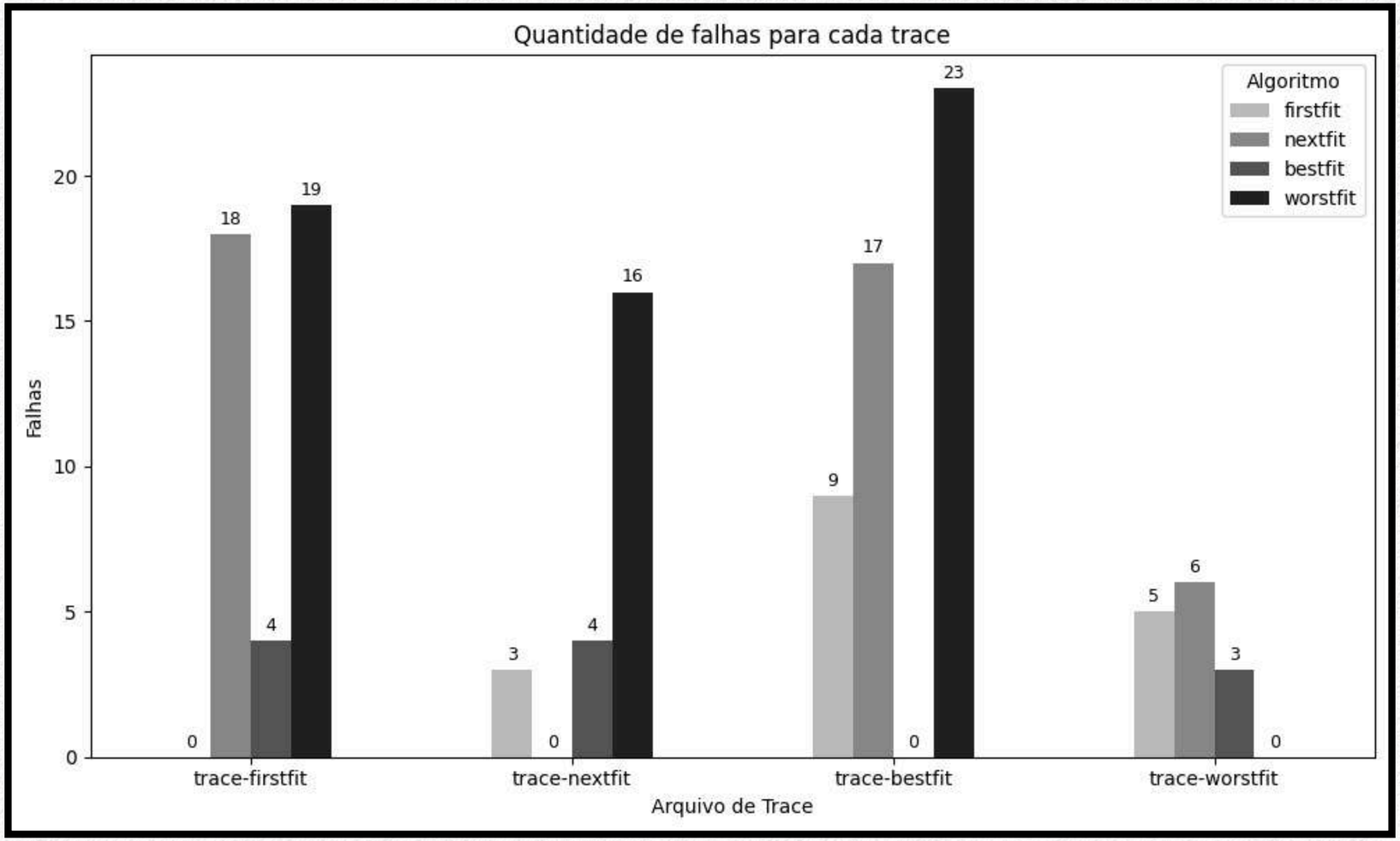
Para cada trace, foram inseridas alterações que fizessem com que um algoritmo específico não falhasse em nenhuma requisição enquanto os demais não conseguissem cumprir todas.

Nenhum trace realizou compactação, visto que o desempenho dos algoritmos ficava muito similar após a compactação, dificultando a comparação.

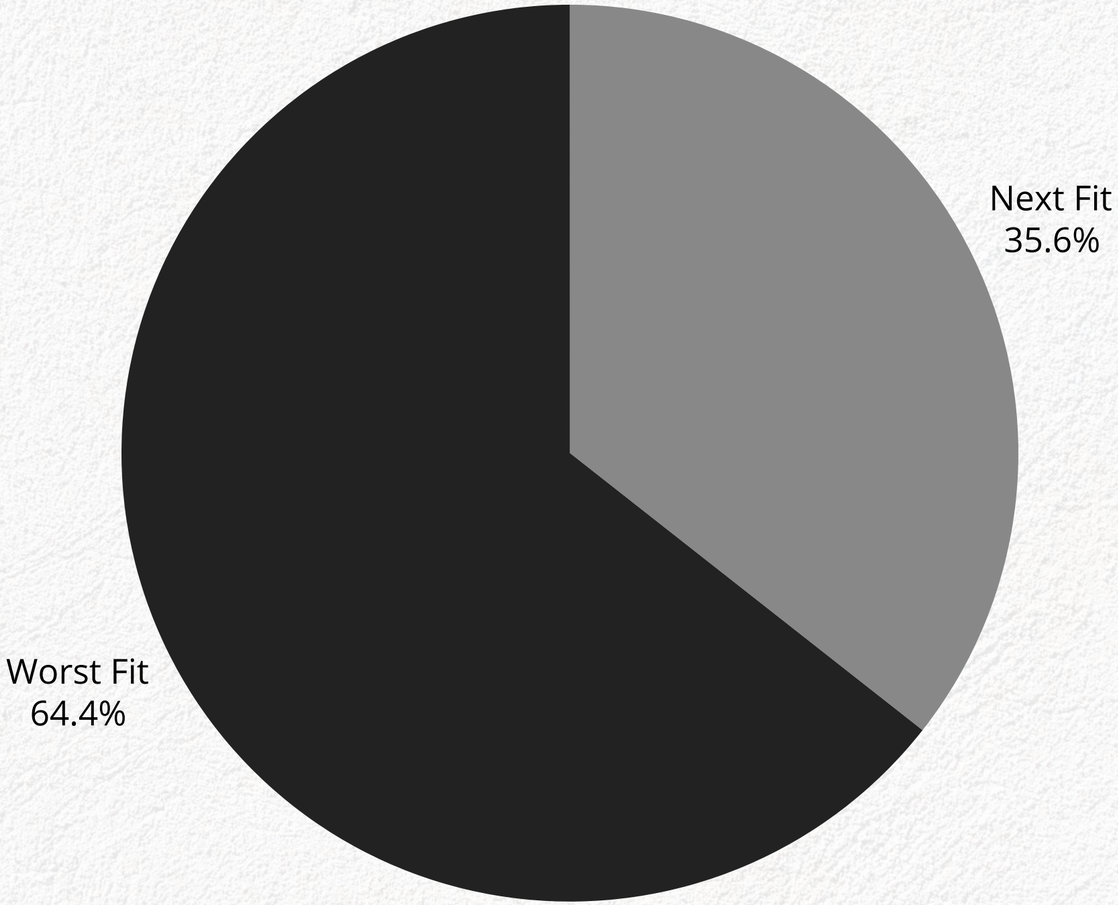
Resultados - Tempo



Resultados - Falhas



Resultados - Menos falhas em trace aleatório



First Fit

- Como podemos observar, em relação ao tempo, o first fit tem um desempenho em média duas vezes mais rápido que os algoritmos que escaneam a memória por completo. Isso condiz com o esperado, já que ele só escaneia a memória até encontrar espaço livre suficiente. Em relação ao next fit, é 3 a 5 vezes mais lento, o que também é esperado já que o Next Fit busca justamente otimizar o First Fit ao não escanear novamente as posições que acabaram de ser preenchidas.
- Em relação a variáveis adicionais, o first fit é o algoritmo menos custoso, não utilizando nenhuma.
- Sua principal desvantagem, aproveitada nos traces dos outros algoritmos, é a de sempre priorizar alocações na região inicial da memória, de modo que ele falhou nas situações em que alocar no começo gerava espaços pequenos inutilizáveis.

Next Fit

- O Next Fit tem um desempenho de tempo expressivamente melhor que os demais algoritmos, chegando a ser mais de 12 vezes mais rápido, o que é esperado justamente por sua estratégia de evitar procurar novamente em posições que acabaram de receber alocações.
- Usou somente uma variável adicional, que indicava a posição onde terminou a última alocação.
- Sua desvantagem principal, explorada pelos traces em que outros algoritmos desempenharam melhor, é a de não alocar pequenos espaços de memória que ficaram para trás nas alocações iniciais. Por exemplo, se os espaços livres fossem de tamanho 10, 20 e 250 e as requisições 25, 10, 20, 225, o Next Fit falharia na última requisição, deixando a memória com posições livres 10, 20 e 195.

Best Fit

- O Best Fit, por escanear toda a memória a cada alocação, tem um tempo de execução bem maior que os dois algoritmos anteriores, sendo essa uma de suas principais desvantagens.
- No entanto, esse scan por toda a memória permite que ele evite fragmentar grandes regiões livres da memória, de modo que grandes requisições feitas depois de várias requisições pequenas ainda poderão ser satisfeitas.
- Ao longo do scan, suas duas variáveis adicionais registravam o tamanho e localização do menor bloco de memória livre disponível para satisfazer aquela requisição.
- Um problema nessa abordagem, explorado pelos arquivos de trace em que os outros algoritmos têm menos falhas, é a de deixar pequenos espaços de memória inutilizáveis, de modo que as falhas dele foram causadas por requisições que vieram depois de outras que geraram esse tipo de fragmentação.

Worst Fit

- O Worst Fit, assim como o Best Fit, tem desempenho de tempo pior por ler toda a memória, enquanto utiliza suas duas variáveis adicionais para armazenar o maior intervalo possível de memória livre para satisfazer aquela requisição.
- No entanto, sua estratégica é antagônica ao algoritmo anterior, buscando evitar deixar os espaços pequenos inutilizáveis. Essa lógica é boa para casos com requisições de tamanho médio seguidas por requisições menores no final. No script que gerava 500 traces com requisições de tamanhos aleatórios entre 1 e 256, o Worst Fit teve o menor número de falhas em cerca de 64% dos casos, mostrando seu bom desempenho para o caso médio.
- No entanto, quando há muitas requisições de tamanho grande no final, como o Worst Fit prioriza alocar primeiro os espaços maiores, ele falha. Por isso, ele foi o que mais falhou nos traces usados para os testes cujos resultados estão expostos nos gráficos anteriores, sendo essa sua principal desvantagem.

Conclusão

- Como era esperado, os algoritmos que fazem o scan completo da memória são muito mais lentos que os outros dois. Portanto, em situações em que a prioridade seja minimizar o overhead do algoritmo de alocação, provavelmente a melhor escolha entre os quatro algoritmos será o next fit.
- Já quando o objetivo é reduzir o número de falhas de memória, observamos que existem situações que maximizam o desempenho de cada um dos algoritmos, sendo relevante fazer uma escolha pensada para cada tipo de caso. Pelos testes adicionais, usando traces gerados aleatoriamente, foi observado que o Worst Fit pode ser o algoritmo ideal a ser escolhido quando não se tem informações suficientes para definir o melhor algoritmo.