DEEC
DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES
TÉCNICO LISBOA

AED - 2022-2023 - 1º Semestre
Algoritmos e Estruturas de Dados

Exemplo de Utilização de Acervos

O objectivo deste código é o estudo e manipulação de informação através de acervos (*heaps*).

Para exemplificar este estudo vamos supor a existência de um mini-sistema operativo multiprocesso que divide a utilização dos seus recursos entre um número limitado de processos "activos" (isto é já iniciados). Existindo apenas uma unidade de processamento, em cada momento apenas um processo pode estar em execução, tipicamente o processo de maior prioridade. Informação sobre os restantes processos já "activos", é guardada numa fila onde os processos estão ordenados pela sua prioridade. Quanto maior a prioridade do processo, maior a sua probabilidade de lhe ser rapidamente dado acesso à unidade de processamento e aos recursos para execução. De forma a permitir a execução atempada de todos os processos, o sistema atribui a cada processo algum tempo de processamento, após o que esse processo é colocado de novo na fila, com uma prioridade mais baixa, de forma a dar oportunidade aos restantes processos na fila de serem executados.

Vamos assumir neste laboratório que a fila de prioridades dos processos do mini-sistemas está implementada através de um acervo.

No código disponibilizado assume-se que podem estar "activos" (instanciados) `N` processos no sistema e os processos são identificados por um *process-id*, `pid` que é um número inteiro entre `0` e `N-1`. A prioridade de cada processo, `prio`, é representada por um número inteiro positivo.

O ficheiro `simproc.c` contém um pequeno programa que permite manipular processos, possibilitando a criação, terminação, execução de um dado processo, etc. Dado que a implementação da fila de prioridades que contém os dados de cada processo em espera é um acervo, o interface do programa permite igualmente experimentar várias operações sobre acervos (até `N` dados). No código disponibilizado é definido que `N=16` (para acervos maiores terá de alterar a definição em `simproc.c`). Utilize a opção "h"do menu do `simproc.c` para verificar as operações disponíveis ou a desenvolver.

Os ficheiros `heap.c` e `heap.h` contêm a implementação e a interface para um tipo `heap` que implementa o acervo. Este tipo de dados está implementado como um tipo abstracto o que lhe confere grande versatilidade. Nesta implementação, o acervo contém elementos do tipo `Item`, definido no ficheiro `Item.h` que é igualmente incluído. Alterando o tipo de dados `Item`, o mesmo código pode ser usado como acervo de outros tipos de dados. No nosso caso o tipo `Item` é o necessário para, para cada processo, guardar o `pid` de cada processo bem como a sua prioridade, `prio`.

Examine cuidadosamente o código contido nos ficheiros `heap.h` e `heap.c`.

1. Compile e execute o programa cliente, `simproc`. Estude o exemplo (opção "e"do menu do `simproc.c`). Qual é a condição de acervo que está a ser utilizada? Verifique no código como é testada a condição de acervo.
2. Apague ("clean") o acervo e depois insira vários processos no acervo. Em termos do simulador do mini-sistema isto é equivalente a criar novos processos indicando para cada um a sua prioridade (o `pid` de cada processo novo obtém-se incrementando o `pid` do último processo criado). Verifique o funcionamento da rotina `FixUp()`. A ordem própria do acervo é mantida?
3. Altere o valor de um elemento do acervo dada a sua posição no mesmo, aumentando e diminuindo sucessivamente o seu valor. Em termos do simulador do mini-sistema isto é equivalente a alterar a sua prioridade (em Unix/Linux isso pode ser feito com o comando `renice`). A ordem é mantida no acervo? Quais são as funções usadas para o efeito?

4. Remova várias vezes o maior elemento do acervo. Isto corresponde na prática ao sistema passar a execução para esse processo. O que é que acontece? Quais são as funções necessárias para obter este resultado?

5. Analise a função `CleanHeap()`, que apaga todos os elementos de um acervo existente libertando a memória alocada. Repare que esta funcionalidade está associada à opção "c"do menu do `client.c`. Assegure-se ainda que na saída do programa toda a memória alocada é libertada.

6. Analise a função `VerifyHeap()`, que verifica se uma tabela representa um acervo, ou seja, satisfaz a condição de acervo. Verifique que esta funcionalidade está associada à opção "v"do menu do `client.c`.

7. Analise a função `Heapify()`, que transforma uma tabela num acervo usando como critério a prioridade de cada processo. A função a desenvolver recebe como argumento uma estrutura Heap, na qual não é garantido que se verifique a condição de acervo. A função deve efectuar as trocas necessárias para garantir que a estrutura representa um acervo. Repare que esta funcionalidade está associada à opção "f"do menu do `client.c`.

8. Analise a função `HeapSort()`, que deve ordena por ordem crescente (da prioridade dos processos) os elementos do acervo (algoritmo heapsort). Repare que esta funcionalidade está associada à opção "s"do menu do `client.c`. Qual é a complexidade do algoritmo de ordenação?

9. Na alínea 3 verificámos o que sucede quando a prioridade de um processo é alterada. Nesta situação foi alterado um processo dado o seu indíce na tabela. Em certas condições é necessário alterar a prioridade de um processo dado apenas o seu `pid`. Isso obriga a procurar o processo na fila e a alterar a sua prioridade. Como esta operação pode ser executada muitas vezes é importante que seja o mais eficiente possível. Como alteraria a estrutura de dados do acervo para tornar esta operação o mais eficiente possível?