Simulació d'un SO — X58154_ca

Aquesta és la l'última part de la pràctica, en què haureu d'implementar la classe `Processador`. Un processador és simplement un contenidor d'objectes de la classe `Programa`. El mètode més important és `executa(int t)`, que fa que executi `t` unitats de temps dels programes que té encuats. En realitat, **una unitat de temps** vol dir l'execució **d'una instrucció** d'un programa (o, si us ho estimeu més, una **línia** d'un programa). Tenint en compte que un processador pot tenir més d'un programa encuat, com gestiona un processador les `t` unitats de temps?
El processador té programes encuats en una cua de prioritat. La prioritat (d'un programa) és un enter més gran que zero: com més gran, més prioritari. Ara bé, pot ser que dos programes diferents en un processador tinguin la mateixa prioritat. En aquest cas, el programa més prioritari de tots dos serà el que tingui l'identificador de programa més gran (els identificadors són únics a cada programa dins d'un processador). Certament, és una manera arbitrària de *trencar l'empat*.
Per exemple, si tenim un programa `P1` amb l'`id = 2` i `prioritat = 100`, un programa `P2` amb l'`id = 4` i `prioritat = 90` i un programa `P3` amb l'`id = 7` i `prioritat = 90`, l'ordre de prioritat serà `P1, P3, P2`. És a dir, el més prioritari serà `P1`, després `P3` i finalment `P2`.
El que fa el processador amb les `t` unitats de temps és el següent: agafa el programa més prioritari i l'executa, almenys, `t` unitats de temps. Pot passar que el programa acabi i no hagi consumit les `t` unitats. En aquest cas, el programa ja ha acabat i el fa fora de la cua, i desencua el següent programa (per prioritat) i mira d'executar-lo les unitats de temps que li queden. Això ho va fent mentre li quedin programes i unitats de temps. Ara bé, també podria ser que el programa hagués consumit totes les unitats de temps i que, tanmateix, no hagi encara acabat. En aquest cas, el processador el que farà és tornar a encuar aquest programa, però li rebaixarà la prioritat en 10 unitats. Per exemple, si tenia prioritat 47, el reencuarà amb prioritat 37. Com que la prioritat sempre ha de ser ≥ 0, si aquesta rebaixa fa que la prioritat sigui negativa, li posarà prioritat 0.
Això farà que a la cua d'un processador hi hagi només programes que encara no han acabat.
Recordem que cada vegada que executem una instrucció, retornarem l'estatus del programa, que indicarà si el programa ha acabat, i per quin motiu.
Us recordem que el registre `STATUS` pot tenir els següents valors:

- -1 Pila plena.
- -2 Divisió per zero.
- -3 Desempilar pila buida.
- -5 Adreça fora de rang
- -8 Error lectura dispositiu.
- -9 Error escriptura dispositiu.
- -10 Programa acabat.

Un programa acaba de manera *normal* si retorna −10, però pot tenir altres tipus d'incidències, i tornarà un estatus diferent de −10 però també negatiu. En aquest cas, el programa també haurà acabat i haurà de sortir fora de la cua de prioritats del processador i no podrà ser reencuat.
Cada vegada que un programa acaba, cal escriure el contingut del seu dispositiu (això ho fem perquè la interfície dels programes ho simulem amb el dispositiu).
Bàsicament, heu d'implementar dos mètodes: el que ja hem esmentat `executa (int t)`, i el mètode `encuaPrograma(Programa& p)`, que, simplement, encua un programa a la cua de prioritats.
Tingueu en compte que la gestió de les prioritats es fa mitjançant la cua de prioritats `PriorityQueue<T>`, que encua per prioritats. Ara bé, per a fer-ho ha de poder comparar dues instàncies del tipus de la cua (en aquest cas, instàncies de la classe `Programa`). Això ho fa assumint que tots els tipus `T` amb què instanciem `PriorityQueue<T>` tenen l'operació
`bool T::compare(const T& t) const`
implementada. Aquesta operació, que ja teníeu definida al problema `X55206` (*Test de la classe Programa*), ara agafa sentit. Si bé en el problema `X55206` no calia que tornés un valor concret, ara sí que cal que torni cert/fals depenent de la definició de prioritat que hem definit anteriorment. Si teniu aquesta funció ben definida, segons el que hem explicat, llavors la gestió de la prioritat és transparent i la fa la classe `PriorityQueue`.
Fixeu-vos que l'enunciat d'aquest exercici ja ofereix uns fitxers que haureu d'utilitzar per a compilar: `Makefile, program.cpp`, apart de les versions `*.old` dels altres fitxers que heu d'acabar d'implementar i `BST.hpp`. A més, **per a fer aquesta part haureu de penjar també les implementacions de les classes** `Dispositiu` `memoriaBST` i `Programa`, a més del fitxer `BST.cpp` que heu fet en els passos anteriors.
Quan pugeu la vostra solució al jutge, només cal que pugeu un tar construït així:

```
tar cf solution.tar memoriaBST.cpp memoriaBST.hpp BST.cpp dispositiu.cpp
dispositiu.hpp programa.cpp programa.hpp processador.cpp processador.hpp
```

Informació del problema

Autoria: PRO1

Generació: 2026-01-25T21:17:34.010Z