Descrierea soluției – Kmalloc

Stud. Bogdan-Cristian Tătăroiu – University of Cambridge

tim că programul va primi o secvență de alocări care pot fi satisfăcute. Un program corect va putea satisface orice permutare a acestei secvențe, fără să necesite informații suplimentare despre alocările viitoare.

O abordare corectă de rezolvare a problemei constă în alocarea zonei de memorie astfel încât poziția aleasă să ne împiedice cât mai puțin în viitor să alocăm alte zone. Spre exemplu, dacă avem disponibile doar 2 zone continue de mărime 5, respectiv 6 și se cere alocarea unei zone de memorie de mărime 2^1, atunci dacă alocăm blocul în interiorul zonei de mărime 5, atunci în viitor nu vom putea răspunde la secvența de alocări de mărimi 2^2, 2^2, 2^0, care ar putea fi satisfăcută dacă am alege blocul de mărime 6.

Vom partiționa intervalele de memorie disponibile în zone de mărime 2^i. Vom menține 63 de liste înlănțuite. În fiecare din aceste liste vom menține adresele de început a tuturor blocurilor de mărime 2^i. Partiționarea aceasta se poate realiza pentru fiecare interval inițial în mai multe moduri, dar acest lucru nu afectează execuția algoritmului. În exemplul din enunț putem avea un bloc de mărime 2^0 la poziția 0, unul de mărime 2^1 la poziția 7 și două de mărime 2^2 la pozițiile 1 și 9. În total vom avea maxim `O(P log P)` astfel de blocuri în aceste liste.

În momentul unei alocări de mărime 2^{size}, dacă avem o adresă în lista corespunzătoare mărimii size, atunci putem afișa și scoate această adresă din listă. Dacă listă corespunzătoare este goală, atunci trebuie să căutăm cel mai mic bloc mai mare decât cel disponibil în acest moment și să îl partiționăm în blocuri mai mici. Nu putem evita spargerea unui bloc în acest moment, iar dacă am alege un bloc mai mare decât cel minim, atunci asta ne-ar împiedica ulterior să răspundem la un query de mărimea respectivă. Pentru o alocare de mărime 2^1, un bloc de mărime 2^4 poate fi spart în un bloc de mărime 2^3, unul de mărime 2^2 și două de mărime 2^1. Unul din cele două blocuri va putea apoi fi afișat ca răspuns.

În cazul cel mai nefavorabil în care pornim cu un singur bloc de mărime 2^{62} și trebuie să răspundem la 500.000 de alocări de mărime 2^0, se observă ca numărul de operații efectuate pe query sunt de forma 62, 1, 2, 1, 3, 1, 2, 1, 4, ... La fiecare query se efectuează o operație, la fiecare 2 query-uri se efectuează încă una, la fiecare 4 query-uri încă una, etc. Numărul total de operații va fi deci de forma M*(1 + ½ + ¼ + ...) = 2*M = O(M)

Complxitate finală: `O(P log P + M)`.

Evaluatorul comisiei va întrerupe execuția programului vostru în momentul în care citește un răspuns invalid (care se suprapune cu memorie rezervată de exemplu) sau un răspuns diferit de cel optim (în momentul în care se **descoperă** că există o secvență validă pe care programul nu o mai poate satisface, nu în momentul în care o alocare eșuează).

Algoritmul descris este cunoscut în literatura de specialitate sub numele **buddy memory allocation**. O variantă îmbunătățită a acestui algoritm este folosit în interiorul kernel-ului **Linux**, sistem de operare cu care ar trebui să vă familiarizați toți.

Referințe culturale:
[1] Brighter Than A Thousand Suns – Iron Maiden
[2] Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb