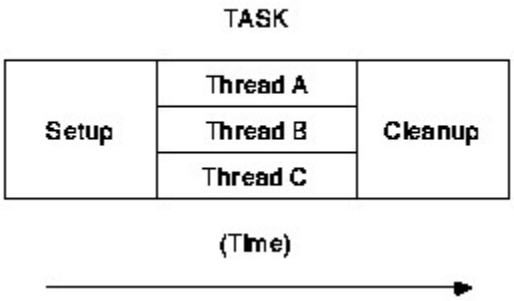
Modelul work-crew

Acest model presupune existenta unui thread de control si a unei sarcini care poate fi impartita in N subsarcini. Threadul de control creeaza un numar de N-1 threaduri, si apoi fiecare thread, inclusiv threadul de control va lucra pe cate o subsarcina. Dupa terminarea lucrului threadul de control asteapta terminarea celorlalte threaduri, colecteaza eventualele rezultate si isi continua executia.



In functie de natura taskului, el poate fi impartit intre threaduri care au aceeasi specializare (exemplu: suma pe coloane a elementelor unei matrice) sau intre threaduri cu specializari diferite (exemplu: suma pe coloanele de rang par ale unei matrice si produsul pe coloanele de rang impar ale unei matrice poate fi rezolvata de threaduri avand cele doua specialzari).

Spre deosebire de modelul boss-workers, threadul de control dupa ce distribuie sarcinile se apuca si el de treaba. Acest lucru este complet justificat, deoarece el nu mai trebuie sa astepte sau sa produca taskuri ca in modelul boss-workers.

Daca s-ar fi creat N threaduri pentru cele N subtaskuri, threadul de control nu ar fi facut altceva decat sa astepte terminarea celor N threaduri. Planificator (scheduler) ar pune acest thread in coada de threaduri blocate pana cand conditia de JOIN ar fi indeplinita, deci threadul nu ar fi consumat timpul procesorului. Singurul neajuns al acestei variante este faptul ca se consuma timp cu crearea unui nou thread (cei drept nu asa de mare ca in cazul crearii unui proces).

In cazul paralelismului datelor, taskul initial este spart in subtaskuri de dimensiuni egale. Ideal ar fi ca numarul de threaduri sa fie egal cu numarul de procesoare si aplicatia sa fie singura care ruleaza pe sistem (adica nici un thread nu poate fi intrerupt pentru a ceda procesorul altuia apartinand altei aplicatii). In acest fel, threadurile ar termina simultan treaba si nu s-ar pierde timp cu asteptarea.

Important: taskurile create sunt complet independente si nu sunt necesare sincronizari intre threaduri cu exceptia JOIN-ului final (threadul de control asteapta terminarea celorlalte threaduri).

Mai multe despre problemele de balansare:

Atunci cand acest model este folosit pentru a imparti un task in mod egal mai multor threaduri, pot aparea probleme de balansare a incarcarii atunci cand impartirea taskului nu a fost facuta in mod judicios. Scopul acestei metode este executia taskului in cel mai scurt timp posibil prin exploatarea la maximum a paralelismului. Cea mai logica impartire a muncii pentru aplicatiile compute-bound este sa se creeze un thread per procesor. Daca aplicatia este I/O bound, trebuie create suficiente threaduri pentru a avea un thread in rulare, sau gata de rulare, pe fiecare procesor, la orice moment dat. Daca un thread se blocheaza in asteptare pentru o resursa I/O, alt thread din aplicatie ar trebui sa fie gata sa ii ia locul (2 threaduri per procesor este un mod de pornire foarte bun).

Intr-o aplicatie compute-bound care doreste sa exploateze paralelismul, schedulerul poate decide sa faca o comutare de context cu unul din threaduri (de exemplu, pentru ca time-slice-ul acestuia a expirat). Sa luam de exemplu o aplicatie care a impartit munca intre 4 threaduri, toate threadurile ruleaza in paralel, si planificatorul (schedulerul) sistemului ii ia resursele unuia dintre threaduri. Pana in momentul in care threadul respectiv primeste din nou procesorul, celelalte 3 threaduri deja si-au terminat munca, si s-au terminat. Al patrulea thread isi face si el munca, si se termina, dar durata intregii operatii a fost cel putin de doua ori mai lunga decat daca toate 4 threadurile ar fi rulat in acelasi timp. Acest scenariu poate fi aplicat oricaruia dintre threadurile worker, putand duce la probleme serioase de balansare si performanta.

Oricare din tehnicile urmatoare poate fi aplicata pentru prevenirea sau minimizarea problemelor de balansare a incarcarii:

- Garantarea faptului ca aplicatia este singurul proces care ruleaza in sistem. Daca nu exista alte aplicatii cu care aplicatia noastra concureaza pentru controlul procesorului, nu poate apare preemption, deci threadurile aplicatiei noastre nu pot pierde procesorul. Din pacate, aceasta nu este in general o solutie realizabila.
- Asignarea threadurilor worker cu atribute de schedulare care le asigura cea mai mare prioritate intre threadurile SCHED_FIFO din sistem. Politica SCHED_FIFO asigura faptul ca nu va avea loc expirarea unui time-slice pentru unul din threadurile worker. Deoarece threadurile worker au cea mai mare prioritate din sistem, ele nu vor fi nici preempted in favoarea altor threaduri din sistem. Deoarece in multe sisteme politicile SCHED_FIFO si SCHED_RR sunt rezervate pentru aplicatiile privilegiate, nici aceasta optiune s-ar putea sa nu fie viabila. In plus, in cazul in care fiecare thread are de prelucrat un task de dimensiuni mari, probabil ca nu este de dorit sa monopolizam procesorul pentru o lunga durata de timp.
- Exista sisteme care ofera o asa numita politica de planificare in grup (gang scheduling policy). In acest caz, un grup de threaduri definit de aplicatie va fi planificat ca un tot unitar. Toate threadurile din gang se executa la acelasi timp si au aceleasi time-slice. Deoarece threadurile sunt planificate impreuna si se executa impreuna, ele se vor termina impreuna (presupunand ca taskul este impartit threadurilor in mod egal). O politica de planificare de gang permite unei aplicatii sa profite in adevaratul sens al cuvantului de paralelismul fizic oferit in sistemele multiprocesor.

Totusi, nu toate sistemele suporta o politica de planificare in grup.

- Optiunea finala pentru a preveni sau a minimiza problemele de balansare care apar este spargerea taskului in bucatele mai mici, si crearea a mai mult de un thread per procesor. La cea mai fina granularitate, fiecare bucatica de task se termina intr-un time-slice. Aceasta va garanta faptul ca fiecare thread isi termina taskul propriu fara a fi time-sliced. Trebuie precizat insa ca overheadul asociat cu crearea, terminarea, si sincronizarea unui numar mare de threaduri pentru taskuri de dimensiuni mici ar putea cauza o depreciere serioasa a performantelor. Cel mai bine este sa experimentati cu numere diferite de threaduri pentru a gasi combinatia ideala intre balansarea incarcarii si performanta. Sugestia noastra este sa porniti cu un thread per procesor, si sa cresteti numarul de threaduri pana cand se atinge balanta potrivita aplicatiei voastre.