Przetwarzanie rozproszone
Projekt

Serwis komputerowy

12 maja 2012

Prowadzący: mgr inż. Michał Kalewski

Autor: **Łukasz Wieczorek** inf94385 gr. I5.2 wieczorek1990@gmail.com

Zajęcia czwartkowe, tygodnie nieparzyste, 11:45

1 Treść zadania

System złożony z K komputerów, które niestety systematycznie wymagają serwisowania. Serwis posiada N stanowisk naprawczych możliwiających jednoczesną naprawę do N maszyn. Przed naprawą każdy komputer musi być przyjęty na dowolnym spośród dostępnych S stanowisk przyjęć, a po naprawie wydawany jest również przez jedno dowolne z nich. Przyjmujemy: K>N>S. Napisać program dla procesu komputera umożliwiający mu systematyczne serwisowanie wg powyższych zasad. Stanowiska przyjęć/wydań oraz naprawcze należy traktować jako zasoby.

2 Przyjęty model komunikacji

Model: przekazywanie komunikatów
Topologia połączeń: klika
Kanały: FIFO

3 Opis protokołu

Protokół bazuje na algorytmie z książki "Podstawy programowania współbieżnego i rozproszonego"Moderachi Ben-Ari, WNT Warszawa 2009, wydanie drugie, s. 209. Algorytm można także przeczytać w oryginalnym dokumencie po angielsku: http://www.ic.unicamp.br/∼buzato/teaching/2011/mo441/carvalho83.pdf s. 3.

Algorytm przekazujący żeton Ricarta-Agrawali

```
boolean mamŻeton <- true w węźle 0, false w pozostałych
integer array[WĘZŁY] żądał <- [0, ..., 0]
integer array[WĘZŁY] dostal <- [0, ..., 0]
integer mójNum <- 0
boolean wSK <- false

prześlijŻeton
  if istnieje N takie, że żądał[N] > dostał[N]
    dla pewnego takiego N
      send(żeton, N, dostał)
      mamŻeton <- false

Główny
  loop forever
    sekcja lokalna
    if not mamŻeton
      mójNum <- mójNum + 1
      dla każdego innego węzła N
        send(żądanie, N, mójID, mójNum)
      receive(żeton, dostał)
      mamŻeton <- true
    wSK <- true
    sekcja krytyczna
    dostał[mójID] <- mójNum
    wSK <- false
    prześlijŻeton
```

```
Odbiorca
  integer nadawca, numerŻądania
  loop forever
    receive(żądanie, nadawca, numerŻądania)
    żądał[nadawca] <- max(żądał[nadawca], numerŻądania)
    if mamŻeton and not wSK
      prześlijŻeton
```

Modyfikacje:

- Zmienne:

  ```
  integer array[S] poczekalnie <- [0, ..., 0]
  integer array[N] serwisy <- [0, ..., 0]
  ```

- Instrukcję `receive(żeton, dostał)` oraz `send(żeton, N, dostał)` zamienić na:

  ```
  receive(żeton, dostał, poczekalnie, serwisy)
  send(żeton, N, dostał, poczekalnie, serwisy)
  ```

- W sekcji lokalnej – obsługa stanów:

  ```
  SPRAWNY, ZEPSUTY, PRZYJMOWANY, NAPRAWIANY, NAPRAWIONY, WYDAWANY
  ```

- W sekcji krytycznej – obsługa stanów:

  ```
  ZEPSUTY, PRZYJMOWANY, NAPRAWIONY, WYDAWANY
  ```

- Uwzględnienie lokalnych sekcji krytycznych dla zmiennych współdzielonych między wątkami Główny i Odbiorca

- Sekwencja stanów i zdarzeń koniecznych do przejścia do następnego stanu:

  ```
  SPRAWNY / zepsuł się ->
  ZEPSUTY / zarezerwował poczekalnię ->
  PRZYJMOWANY / zarezerwował serwis, zwolnił poczekalnię ->
  NAPRAWIANY / naprawiony ->
  NAPRAWIONY / zarezerwował poczekalnię, zwolnił serwis ->
  WYDAWANY / zwolnił poczekalnię -> SPRAWNY
  ```

- Komputer nie jest "przyjmowany"(przeciwieństwo "wydawany") do poczekalni jeżeli w poczekalni jest tylko jedno miejsce oraz wszystkie stanowiska serwisowe są zajęte

Operacje lock(zmienna1, ...), unlock(zmienna1, ...) oznaczają założenie zamków odpowiadających za wyłączny dostęp do daneych zmiennych. Algorytm po modyfikacji:

```
integer mojID <- identyfikator procesu
boolean mamŻeton <- true w węźle 0, false w pozostałych
integer array[K] żądał <- [0, ..., 0]
integer array[K] dostal <- [0, ..., 0]
integer array[S] poczekalnie <- [0, ..., 0]
integer array[N] serwisy <- [0, ..., 0]
integer mójNum <- 0
boolean wSK <- false
enum stan_t = { SPRAWNY, ZEPSUTY, PRZYJMOWANY, NAPRAWIANY, NAPRAWIONY, WYDAWANY }
enum stan_t stan <- SPRAWNY
```

```
prześlijŻeton
  integer array[K+S+N] dane
  lock(żądał, dostał)
  if istnieje N takie, że żądał[N] > dostał[N] oraz N <> mojID
    unlock(żądał)
    dopisz dostał do dane
    unlock(dostał)
    dopisz poczekalnie, serwisy do dane
    dla pewnego takiego N
      send(żeton, N, dane)
      mamŻeton <- false
  else
    unlock(ządał, dostał)

Główny
  integer array[K+S+N] dane
  loop forever
    sekcja lokalna
    lock(mamŻeton)
      if not mamŻeton
        mójNum <- mójNum + 1
        dla każdego innego węzła N
          send(żądanie, N, mójID, mójNum)
        receive(żeton, dane)
        lock(dostał)
          przepisz dostał z dane
        unlock(dostał)
        przepisz poczekalnie, serwisy z dane
        mamŻeton <- true
      lock(wSK)
        wSK <- true
      unlock(wSK)
    unlock(mamŻeton)
    sekcja krytyczna
    lock(dostał)
      dostał[mójID] <- mójNum
    unlock(dostał)
    lock(mamŻeton)
      lock(wSK)
        wSK <- false
      unlock(wSK)
      prześlijŻeton
    unlock(mamŻeton)

Odbiorca
  integer nadawca, numerŻądania
  loop forever
    receive(żądanie, nadawca, numerŻądania)
    lock(żądał)
      żądał[nadawca] <- max(żądał[nadawca], numerŻądania)
    unlock(żądał)
    lock(mamŻeton)
      lock(wSK)
        if mamŻeton and not wSK
          prześlijŻeton
```

```
      unlock(wSK)
    unlock(mamŻeton)

sekcja lokalna
  case (stan)
    SPRAWNY:
      pracuj aż się nie zepsujesz
      zmień stan na ZEPSUTY
    ZEPSUTY:
      czekaj na przyjęcie do poczekalni
    PRZYJMOWANY:
      czekaj na przyjęcie do serwisu
    NAPRAWIANY:
      czekaj aż zostaniesz naprawiony
      zmień stan na NAPRAWIONY
    NAPRAWIONY:
      czekaj na przyjęcie do poczekalni
    WYDAWANY:
      czekaj na wydanie z poczekalni

sekcja krytyczna
  case (stan)
    ZEPSUTY:
      if udało się zarezerwować poczekalnię w celu przyjęcia
        zmień stan na PRZYJMOWANY
    PRZYJMOWANY:
      if udało się zarezerwować serwis
        zwolnij poczekalnię
        zmień stan na NAPRAWIANY
    NAPRAWIONY:
      if udało się zarezerwować poczekalnię w celu wydania
        zwolnij serwis
        zmień stan na WYDAWANY
    WYDAWANY:
      zwolnij poczekalnię
      zmień stan na SPRAWNY

zarezerwuj poczekalnię
  if (jest jedna wolna poczekalnia) and (wszystkie serwisy są zajęte) and (komputer jest przyjmowany)
    return nie udało się zarezerwować poczekalni
  else
    if są wolne poczekalnie
      oznacz dowolną wolną poczekalnię jako zarezerwowaną

zarezerwuj serwis
  if są wolne serwisy
    oznacz dowolny wolny serwis jako zarezerwowany
```

4 Złożoność komunikacyjna

- Czasowa: 2
  Komunikacje: Żądanie, Żeton.

- Komunikacyjna:

– Pakietowa: $(K-1)+1=K$
Żadania wysyłamy do wszystkich, żeton przekazywany jednemu procesowi.

– Bitowa : $32*((K-1)+K+S+N)=32*(2K+S+N-1)$
Abstrahuję od implementacji OpenMPI
32 – rozmiar typu danych MPI_INT w bitach
$K-1$ – ilość żądań
K – rozmiar tablicy dostał w pakiecie żetonu
S, N – rozmiar tablic poczekalnie, serwisy

5 Opis implementacji

Pliki źródłowe:
main.c – główny plik programu
run.sh – skrypt uruchamiający program i prezentujący logi
stop.sh – zatrzymuje procesy
compile.sh – kompiluje program
openmpi-install.sh – kompiluje i instaluje openmpi w wersji dla wielu wątków
clean.sh – usuwa wygenerowane pliki wyjściowe

Obsługa:

```
$ cd PR
$ ./run.sh
ERROR:  Argumenty wywołania: K, N, S.
$ ./run.sh 3 2 1
...
```

Program wyświetli logi wraz ze znaczkami czasowymi oraz zapisze je w odpowiednich plikach log.dd.
Po zatrzymaniu programu (CTRL + C) zaprezetnowane zostaną połączone kolorowane logi.

Pliki wyjściowe:
log.dd – log z maszyny o numerze dd
log – log łączony
out – log w czytelnej formie z kolorami