1

Programowanie wspóªbie»ne

wiczenia 1 problem wzajemnego wykluczania

Zadanie 1

Uruchamiamy wspóªbie»nie dwa nast¦puj¡ce procesy.

int y = 0;

process P1() {

int x, i ;

for (i = 1; i < 6; i++) { x = y;

x = x + 1;

y = x;

}

}

process P2() {

int x, i ;

for (i = 1; i < 6; i++) { x = y;

x = x + 1;

y = x;

}

}

Jak¡ warto±¢ b¦dzie miaªa zmienna y po zako«czeniu dziaªania obu procesów? Przyjmujemy, »e zmienna zadeklarowana poza tre±ci¡ procesu jest zmienn¡ globaln¡, do której dost¦p maj¡ wszystkie procesy.

Zadanie 2

Uruchamiamy wspóªbie»nie dwa nast¦puj¡ce procesy.

process P1() {

while (true) {

sekcja\_lokalna;

protokol\_wstepny; sekcja\_krytyczna; protokol\_koncowy }

}

process P2() {

while (true) {

sekcja\_lokalna;

protokol\_wstepny; sekcja\_krytyczna; protokol\_koncowy }

}

Chcemy zapewni¢, »e w tym samym czasie co najwy»ej jeden z nich wykonuje fragment programu oznaczony jako sekcja\_krytyczna. Jakie instrukcje nale»y umie±ci¢ w protokoªach, aby zrealizowa¢ ten cel? Zakªadamy, »e nie dysponujemy »adnymi mechanizmami synchronizacyjnymi, wi¦c pro tokoªy powinny umiej¦tnie wykorzysta¢ zmienne globalne oraz instrukcje j¦zyka programowania.

*•* Czy protokoªy wst¦pne i ko«cowe mo»na pozostawi¢ puste?

*•* Co oznacza bezpiecze«stwo w przypadku tak sformuªowanego zadania? *•* Co oznacza »ywotno±¢?

W rozwi¡zaniu b¦dziemy korzysta¢ ze zmiennych globalnych i lokalnych. Zmienna lokalna mo»e znajdowa¢ si¦ w prywatnej przestrzeni adresowej procesu. Pozostaªe procesy nie maj¡ do niej dost¦pu, nie mog¡ jej zatem ani odczytywa¢ ani mody kowa¢. Inaczej sytuacja wygl¡da ze zmiennymi globalnymi. S¡ one wspóªdzielone mi¦dzy procesami, co oznacza, »e w dowolnej chwili ka»dy z nich mo»e takie zmienne zmody kowa¢ lub je odczytywa¢.

*•* Co dzieje si¦, gdy dwa wykonuj¡ce si¦ równolegle procesy w tej samej chwili chc¡ uzyska¢ dost¦p do tej samej zmiennej, a zatem do tej samej komórki (tych samych komórek) pami¦ci?

2

Kon ikt rozwi¡zuje sprz¦t za pomoc¡ arbitra pami¦ci. Jest to ukªad sprz¦towy, który realizuje wzajemne wykluczanie przy dost¦pie do pojedynczych komórek pami¦ci. Jednoczesne odwoªania do tej samej komórki pami¦ci zostan¡ w jaki± nieznany z góry sposób uporz¡dkowane w czasie i wykonane. W dalszej cz¦±ci rozwa»a« zakªadamy istnienie arbitra pami¦ci i jego poprawne dziaªanie.

Pierwsza próba rozwi¡zania

Spróbujmy najpierw rozwi¡za¢ problem wprowadzaj¡c zmienn¡ globaln¡ ktoczeka. B¦dzie ona przyjmowa¢ warto±ci 1 lub 2. Warto±¢ 1 oznacza, »e proces pierwszy musi poczeka¢ a prawo wej±cia do sekcji krytycznej ma proces drugi.

Oczekiwanie na wej±cie realizujemy poprzez pust¡ p¦tl¦, której jedynym zadaniem jest cykliczne sprawdzanie warunku na wej±cie do sekcji krytycznej. Proces, który skorzysta z sekcji krytycznej, przekazuje pierwsze«stwo rywalowi.

int ktoczeka = 1;

process P1() {

while (true) {

sekcja\_lokalna;

while (ktoczeka == 1) { } sekcja\_krytyczna;

ktoczeka = 1;

}

}

process P2() {

while (true) {

sekcja\_lokalna;

while (ktoczeka == 2) { } sekcja\_krytyczna;

ktoczeka = 2;

}

}

Zanim przyst¡pimy do analizy poprawno±ci rozwi¡zania, przypomnijmy zaªo»enia dotycz¡ce sek cji krytycznej, sekcji lokalnej i systemu operacyjnego, który nadzoruje wykonywanie procesów. Rozstrzygnij, które stwierdzenia s¡ prawdziwe:

*•* Formuªuj¡c problem wzajemnego wykluczania zakªada si¦, »e:

Ka»dy proces, który wszedª do sekcji krytycznej w sko«czonym czasie j¡ opu±ci. Proces mo»e przebywa¢ w sekcji krytycznej nie dªu»ej ni» ustalony z góry czas. Proces nie mo»e zako«czy¢ si¦ w sekcji krytycznej.

Proces mo»e si¦ zako«czy¢ w sekcji krytycznej ale tylko w sposób poprawny. *•* Proces, który rozpocz¡ª wykonywanie sekcji lokalnej:

musi w sko«czonym czasie zako«czy¢ j¡ i przej±¢ do protokoªu wst¦pnego. mo»e zako«czy¢ swoje dziaªanie na skutek bª¦du.

mo»e zako«czy¢ swoje dziaªanie, ale tylko w sposób poprawny.

Czy wobec przyj¦tych zaªo»e« zaproponowane rozwi¡zanie poprawne? Wªasno±¢ bezpiecze«- stwa jest zachowana nigdy oba procesy nie b¦d¡ jednocze±nie w sekcji krytycznej. A co z wªasno±ci¡ »ywotno±ci? Przypomnijmy o zaªo»eniu, »e proces nie przebywa niesko«czenie dªugo w rejonie krytycznym. Zatem po wyj±ciu z niego (które na pewno w ko«cu nast¡pi) rozpocznie si¦ wykonanie sekcji lokalnej. I tu pojawia si¦ problem, bo o tym fragmencie programu nic zaªo»y¢ nie mo»emy. Je±li proces utknie w tym fragmencie kodu (bo nast¡pi bª¡d, zap¦tlenie, itp.) to drugi z procesów b¦dzie mógª wej±¢ do sekcji krytycznej jeszcze co najwy»ej raz. Kolejna próba zako«- czy si¦ wstrzymaniem procesu na zawsze . Zatem przedstawione rozwi¡zanie nie ma wªasno±ci »ywotno±ci. Inn¡ wad¡ tego rozwi¡zania jest zbyt ±cisªe powi¡zanie ze sob¡ procesów. Musz¡ one korzysta¢ z sekcji krytycznej naprzemiennie, a przecie» potrzeby obu procesów mog¡ by¢ ró»ne. Je±li ponadto dziaªaj¡ one z ró»n¡ szybko±ci¡ (bo na przykªad sekcja lokalna jednego z nich jest bardziej zªo»ona od sekcji lokalnej drugiego), to proces szybszy b¦dzie równaª tempo pracy do wolniejszego.

3

Druga próba rozwi¡zania

Spróbujmy podej±¢ do problemu w inny sposób. Wprowad¹my dwie logiczne zmienne globalne: jest1 oznaczaj¡c¡, »e proces P1 jest w sekcji krytycznej i analogiczn¡ zmienn¡ jest2 dla procesu P2. Przed wej±ciem do sekcji krytycznej proces sprawdza, czy jego rywal jest ju» w sekcji krytycznej. Je±li tak, to czeka. Gdy sekcja krytyczna si¦ zwolni to proces ustawi swoj¡ zmienn¡ na true sygnalizuj¡c, »e jest w sekcji krytycznej, po czym wejdzie do niego.

bool jest1 = false;

bool jest2 = false;

process P1() {

while (true) {

sekcja\_lokalna;

while (jest2) { } jest1 = true;

sekcja\_krytyczna; jest1 = false;

}

}

process P2() {

while (true) {

sekcja\_lokalna;

while (jest1) { } jest2 = true;

sekcja\_krytyczna; jest2 = false;

}

}

Zauwa»my, »e to rozwi¡zanie nie uzale»nia ju» procesów od siebie. Je±li jeden z nich nie chce korzysta¢ z sekcji krytycznej lub awaryjnie zako«czy swoje dziaªanie w sekcji lokalnej, to drugi mo»e swobodnie wchodzi¢ do sekcji krytycznej ile razy zechce. Nie ma te» problemu z »ywotno±ci¡. Je±li pierwszy proces utkn¡ª w p¦tli w protokole wst¦pnym, to drugi proces musi znajdowa¢ si¦ gdzie± mi¦dzy przypisaniem jest2 na true a przypisaniem jest2 na false. Po sko«czonym czasie wyjdzie zatem z rejonu krytycznego i ustawi zmienn¡ jest2 na false pozwalaj¡c partnerowi wyj±¢ z jaªowej p¦tli i wej±¢ do rejonu krytycznego. Niestety, przy pewnych zªo±liwych przeplotach mo»e si¦ zdarzy¢, »e do rejonu krytycznego wejd¡ oba procesy. Wska» taki scenariusz.

Przyczyn¡ takiej sytuacji jest zbyt pó¹ne ustawienie zmiennych logicznych. Proces ju» prawie jest w sekcji krytycznej (bo przeszedª przez wstrzymuj¡c¡ go p¦tl¦), a jeszcze nie poinformowaª o tym rywala (to jest nie ustawiª swojej zmiennej jest).

Poniewa» istnieje przeplot, który prowadzi do bª¦dnej sytuacji, wi¦c zgodnie z de nicj¡ po prawno±ci (musi by¢ dobrze dla ka»dego przeplotu) stwierdzamy, »e powy»szy program jest niepoprawny. Zauwa»my jednak, »e gdyby sprawdzenie warunku w p¦tli oraz zmiana warto±ci zmiennej byªy niepodzielne, to takiego zªego scenariusza nie daªoby si¦ pokaza¢. Tyle tylko, »e zagwarantowanie niepodzielno±ci oznacza stworzenie sekcji krytycznej, a to jest wªa±nie problem, który rozwi¡zujemy.

Trzecia próba rozwi¡zania

Poniewa» zmiana warto±ci zmiennych globalnych w poprzednim rozwi¡zaniu zostaªa dokonana zbyt pó¹no, wi¦c spróbujmy zmieni¢ kolejno±¢ instrukcji i ustawmy najpierw zmienne logiczne, a potem dopiero próbujmy przej±¢ przez p¦tle. Teraz zmienne logiczne oznaczaj¡ ch¦¢ wej±cia do sekcji krytycznej.

4

bool chce1 = false; bool chce2 = false;

process P1() {

while (true) {

sekcja\_lokalna;

chce1 = true;

while (chce2) { }

sekcja\_krytyczna;

chce1 = false;

}

}

process P2() {

while (true) {

sekcja\_lokalna;

chce2 = true;

while (chce1) { } sekcja\_krytyczna; chce2 = false;

}

}

Teraz mamy program bezpieczny. Faktycznie w rejonie krytycznym mo»e znajdowa¢ si¦ co naj wy»ej jeden proces. Ale brakuje »ywotno±ci! Z ªatwo±ci¡ mo»na doprowadzi¢ do zakleszczenia. Wska» odpowiedni scenariusz.

Czwarta próba rozwi¡zania

Mo»na próbowa¢ ratowa¢ sytuacj¦ zmuszaj¡c procesy do chwilowej rezygnacji z wej±cia do sekcji i ust¡pienia pierwsze«stwa.

bool chce1 = false;

bool chce2 = false;

process P1() {

while (true) {

sekcja\_lokalna;

chce1 = true;

while (chce2) {

chce1 = false;

chce1 = true;

}

sekcja\_krytyczna; chce1 = false;

}

}

process P2() {

while (true) {

sekcja\_lokalna;

chce2 = true;

while (chce1) {

chce2 = false;

chce2 = true;

}

sekcja\_krytyczna; chce2 = false;

}

}

Niestety znów istnieje zªo±liwy przeplot, który powoduje brak »ywotno±ci. Nie pomo»e argumen tacja, »e taki przeplot jest w zasadzie nieprawdopodobny. Zgodnie z de nicj¡ poprawno±ci, skoro istnieje scenariusz powoduj¡cy brak »ywotno±ci, to program jest niepoprawny.

Algorytm Petersona

Poprawne rozwi¡zanie znane pod nazw¡ algorytmu Petersona jest poª¡czeniem pierwszego pomysªu z przedostatnim. Utrzymujemy zmienne chce1 i chce2, które oznaczaj¡ ch¦¢ wej±cia procesu do sekcji krytycznej. Je±li obydwa procesy chc¡ wej±¢ do sekcji krytycznej rozstrzygamy kon ikt za pomoc¡ zmiennej ktoczeka.

5

bool chce1 = false;

bool chce2 = false;

int ktoczeka = 1;

process P1() { while (true) { sekcja\_lokalna; chce1 = true;

ktoczeka = 1;

process P2() { while (true) { sekcja\_lokalna; chce2 = true;

ktoczeka = 2;

while (chce2 && (ktoczeka == 1)) { } sekcja\_krytyczna;

chce1 = false;

}

}

}

while (chce1 && (ktoczeka == 2)) { } sekcja\_krytyczna;

chce2 = false;

}

Przypomnijmy, »e nie zakªadamy atomowo±ci (niepodzielno±ci) poszczególnych instrukcji (w szcze gólno±ci wyliczanie zªo»onych warunków logicznych nie musi odbywa¢ si¦ atomowo). Zakªadamy natomiast istnienie arbitra pami¦ci (mo»e doj±¢ do jednoczesnej próby zmiany warto±ci zmiennej ktoczeka).

Zauwa»my, »e w sytuacji, gdy tylko jeden proces rywalizuje o dost¦p do sekcji krytycznej, to b¦dzie on mógª do woli z niej korzysta¢, bo zmienna chce jego rywala ma caªy czas warto±¢ false. Dzi¦ki temu unikamy ±cisªego powi¡zania procesów, jak byªo to w rozwi¡zaniu pierwszym. Z drugiej strony, gdy oba procesy ci¡gle chc¡ korzysta¢ z sekcji krytycznej, robi¡ to na zmian¦.

Analiza algorytmu

Przeanalizuj algorytm Petersona pod k¡tem zmiany kolejno±ci wykonywanych instrukcji: *•* Czy mo»na zamieni¢ kolejno±¢ przypisa« przed p¦tl¡ while?

*•* Czy mo»na umie±ci¢ zmian¦ warto±ci zmiennej ktoczeka po wyj±ciu z sekcji krytycznej? *•* Czy mo»na zamieni¢ kolejno±¢ sprawdzania warunków w p¦tli while?

*•* Czy mo»na zainicjowa¢ zmienn¡ ktoczeka na inne warto±ci?

*•* A zmienne chce1 i chce2?

Wady algorytmu Petersona:

*•* Aktywne oczekiwanie. Z zaªo»enia nie byªo dost¦pnych »adnych mechanizmów wstrzymy wania procesów, nie pozostaªo nic innego ni» zatrzymanie procesu w jaªowej p¦tli. Anga»uje to jednak czas procesora (i szyn¦ danych). W przyszªo±ci aktywne oczekiwanie b¦dziemy traktowa¢ jak powa»ny bª¡d.

*•* Liczba procesów musi by¢ znana z góry.

*•* Koszt protokoªu wst¦pnego jest znaczny.