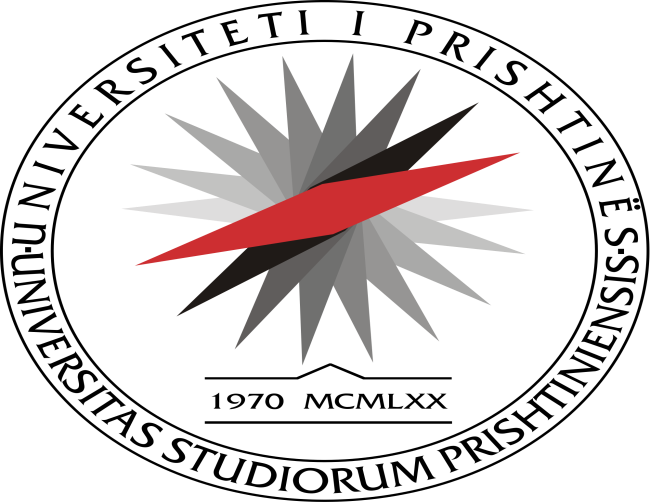
**UNIVERSITETI I PRISHTINËS ”HASAN PRISHTINA”**

**FAKULTETI I INXHINIERISË ELEKTRIKE DHE KOMPJUTERIKE**

****

**Detyra :** Algoritmi i Petersonit për sinkronizim të proceseve

**Lënda:** Sisteme Operative **Mentori**:Mr.Sc Artan Mazrekaj

**Studentët:** Edon Nura

Diana Balinca 20/01/2017

Përmbajtja

1. **Hyrje3**
   1. Qëllimi i dokumentit3
   2. Definimi i termave dhe shkurtesave3
   3. Referencat3
2. **Përshkrimi i përgjitshëm i detyrës3**
   1. Algoritmi i Petersonit3
   2. Përparësitë e algoritmit4
   3. Dobësitë e algoritmit4
3. **Përshkrimi i hollësishëm i detyrës 4**
   1. Ekzekutimi para hyrjes në critical section5
   2. Critical section6
   3. Ekzekutimi pas daljes nga critical section6

4. **Testimi6**

**1. Hyrje**

**1.1. Qëllimi i dokumentit**

Qëllimi i këtij dokumenti është përshkrimi i detyrës së parashtruar, përkatësisht metodave dhe teknikave që janë përdorur në realizimin e suksesshëm të saj.

**1.2. Definimi i termave dhe shkurtesave**

**Definimi i shkurtesave**

CS Critical section

**Definimi i termave**

* **Mutual exclusion** (mutex) është një object program i cili pengon qasjen e njëkohësishmë në një resurs të përbashkët.
* **Critical section** është pjesa e kodit që ofron qasjen tek resursi i përbashkët.
* **Thread** është një numër i instruksioneve të programuara të cilat mund të menaxhohen në mënyrë të pavarur nga sistemi operativ.
* **Resurset e përbashkëta** janë resurse e qasshme nga tw gjitha proceset si fajllat, variablat etj.

**1.3.Referencat**

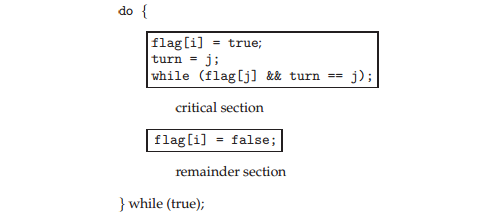
* [*https://en.wikipedia.org/wiki/Peterson's\_algorithm*](https://en.wikipedia.org/wiki/Peterson's_algorithm)
* [*http://www.geeksforgeeks.org/petersons-algorithm-for-mutual-exclusion-set-1/*](http://www.geeksforgeeks.org/petersons-algorithm-for-mutual-exclusion-set-1/)

**2. Përshkrimi i përgjithshëm i detyrës**

**2.1. Algoritmi i Petersonit**

Algoritmi i Petersonit është algoritëm i zgjidhjes së situatave në të cilat dy a po më shumë procese kërkojnë të shfrytëzojnë në të njejtën kohë resurset e përbashkëta(critical section), zgjidhja e këtyre situatave mundësohet përmes komunikimit me memorien e përbashkët. Ky algoritëm është formuluar nga Gary L. Peterson në vitin 1981, versioni origjinal i algoritmit është formuluar që të zgjedh situatat vetëm në mes të dy proceseve por ai mund të përgjithësohet për më shumë se dy procese.

Fig 2.1.1



Algoritmi përdor dy variabla, **flag** dhe **turn**. Vlera true e variablës **flag[n]** tregon se procesi **n** dëshiron të hyjë në critical section. Hyrja në critical section e procesit **P0** është e mundësuar

në qoftë se procesi **P1** nuk dëshiron të hyjë në këtë seksion ose **P0** ka prioritet ndaj **P1**. Algoritmi plotëson tri kushte thelbësore për zgjidhjen e critical section siq janë mutual exclusion, progress dhe bounded waiting.

**Mutual exclusion** është kushti i cili iu pamundëson dy proceseve P0 dhe P1 që në të njejtën kohë të jenë në critical section, plotësimi i këtij kushti arrihet përmes dy variablave të lartëcekura. Kur P0 është në critical section atëherë flag[0] ka vlerën true, ndërsa flag[1] ka vlerën false, që i bie se P1 sapo ka lëshuar critical section ose turn ka vlerën 0, që i bie se P1 tenton të futet në critical section.

**Progress** është kushti i cili thotë se në qoftë se asnjë proces nuk është në critical section dhe ka procese të atilla që deshirojnë të futën në këtë seksion, atëherë vetëm proceset që nuk janë duke u ekzekutuar në seksionet tjera të tyre mund të kontribojnë në zgjedhjen e procesit të radhës i cili do të futet ne cs.

**Bounded waiting (bounded baypass)** do të thotë se numri i përparësive për të hyrë në critical section i proceseve tjera në raport me një process të caktuar i cili ka qëllim të hyjë në këtë seksion është i kufizuar nga numri i proceseve në sistem. Asnjë proces në algoritmin e Petersonit nuk pret më shumë se një radhë për hyrje në cs.

**2.2. Përparësitë e algoritmit të Petersonit**

Një nga karakteristikat më kryesore e algoritmit është thjeshtësia, thjeshtësia e algoritmeve do të thotë shpejtësi më e madhe e ekzekutimit të tyre, gjithashtu zgjidhja e Petersonit paraqet një metodë mjaft efikase për menjanimin e mutual exclusion tek sistemet me një njësi procesorike.

**2.3. Dobësitë e algoritmit**

Algoritmi i Petersonit është zhvilluar në vitet e hershme të zhvillimit të kompjuterit dhe si i tillë është menduar për një arkitekturë shumë më të thjeshtë kompjuterike. Në sistemet moderne kompjuterike mënyra se si performohen instruksionet bazike të makinës si thirrja dhe ruajtja e programeve nuk siguron që algoritmi do të punojë me saktësi. Zhvillimi i kompjuterëve me më shumë bërthama procesorike është një sfidë tjetër për algoritmin, pasi që nga një resurs i përbashkët për tu ndarë mes proceseve tani kemi disa dhe ky algoritëm nuk parashah zgjedhjen e sitatave të këtilla.

**3. Përshkrimi i hollësishëm i detyrës**

Në programin kryesor main bëhet deklarimi i variblave të tipit **pthreat\_t** në të cilat ruhet ID e thread-ave që do të krijohen në vijim, thread-at krijohen me funksionin pthread\_create() i cili ka disa parametra. Parametrat e funksionit janë sin ë vijim:

1. Është pointer tek një pthread variabël në të cilën ruhet ID e thread-it të krijuar.
2. Është pointer tek thread attribute dhe kontrolon detajet se si thread-i ndërvepron me pjesën tjetër të programit, vlera NULL krijon një thread me atribute të paracaktuara.
3. Tregon tek një thread function.
4. void\*  (\*)  (void\*

Funksioni i thirrur në vazhdim pthread\_join() ndalon ekzekutimin e thirrjes se threadit deri sa

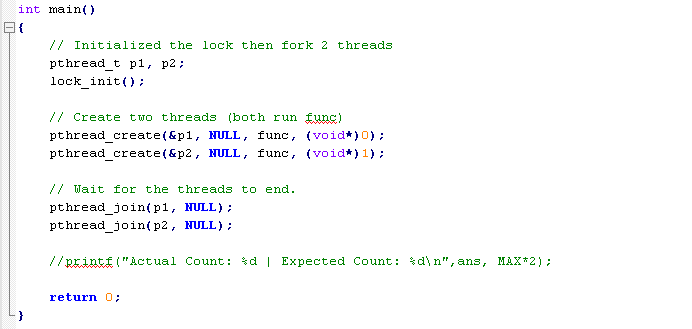
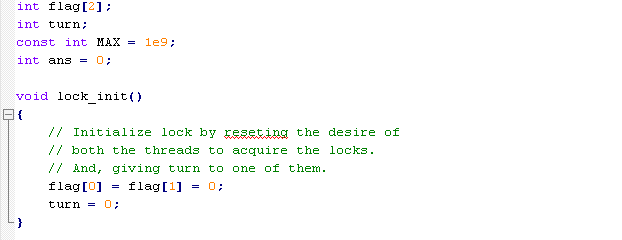


Fig.3.1

**3.1. Ekzekutimi para hyrjes në critical section**

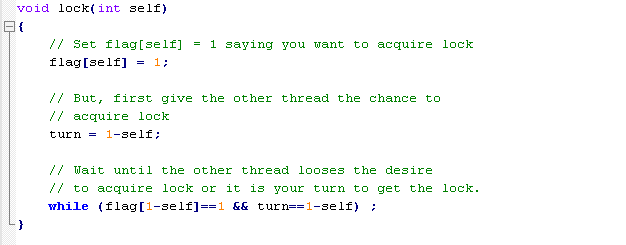
Fillimisht bëhet deklarimi i variablave **flag** dhe **turn** të nevojshme për implementimin e algoritmit të Petersonit, dhe ato variabla inicializohen me vlerën 0(false) si variabla të cilat nuk kanë tendencë për të hyrë në critical section.

Fig.3.1.1



Para hyrjes ne critical section duhet që proceseve të ju japim qëllimin e futjes në këtë seksion, kjo bëhet me inicializimin e variablës flag me vlerë ‘true’ apo 1, variabla tjetër turn shërben në rastet kur dy procese kan qëllim të futen në cs në të njejtën kohë atherë ndrrimi i vlerës së kësaj variable në 0 i jep prioritet procesit P0 ndërsa vlera e saj 1 i jep prioritet procesit P1.Unaza ëhile pret përderisa ndonjëri prej proceseve humb dëshiren për të hyrë në CS apo njërit proces i jepet përparësi në raport me procesin tjetër dhe kalimi i kushtin të saj paraqet edhehyrjen në critical section.

Fig 3.1.2



**3.2. Critical section**

Ky seksion paraqet pjesën e kodit e cila i iu ofron qasje proceseve(thread-ve) në resurset e përbashkëta, dhe vetëm një thread mund të hyjë në këtë seksion në të njejtën kohë.

Në këtë funksion është simuluar një detyrë e thread-it për incrementimin e një variable ans

nga 0 në vlerën maximale MAX të definuar më lartë.

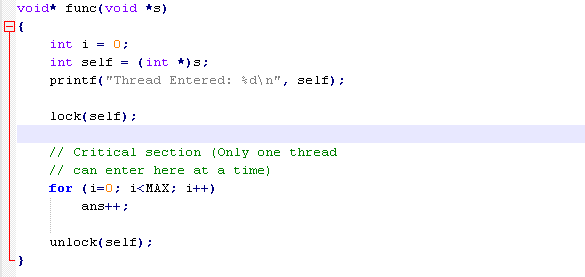


Fig 3.2.1

**3.3. Ekzekutimi pas daljes nga critical section**

Pas përfundimit të shfrytëzimit të resurseve bëhet riinicializimi i variablës flag me vlerën 0(false) në këtë mënyrë ia heqim thread-it qëllimin për tu future përsëri në critical section.

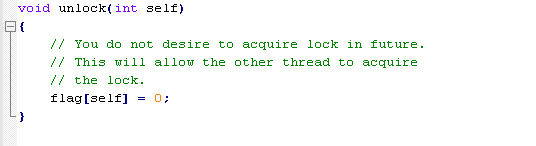


Fig 3.3.1

**4.Testimi**

Fajlli që përmban kodin burimor të programit të zhvilluar me algoritmin e Petersonit për tu ekzekutuar duhet fillimisht duhet të kompajllohet me anë të komandës

**gcc -pthread emrifajllit.c -o fajllikompajlluar**

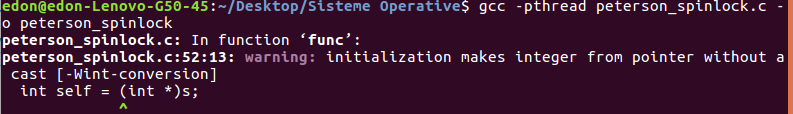
****

Fig 4.1

Ekzekutimi i suksesshëm i programit shfaq rezultation si ne Fig 4.2.

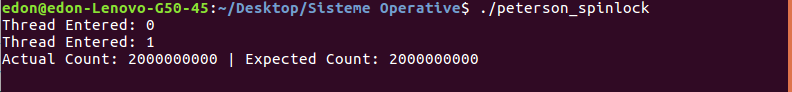
****

Fig 4.2