Λειτουργικά Συστήματα

2021 - 2022

1η Εργαστηριακή Άσκηση

Μέρος 1 [30 μονάδες]

Ερώτημα Α [10]: Εξηγήστε προσεκτικά τι κάνουν τα παρακάτω προγράμματα:

```
#include <unistd.h>
#include <stdio.h>
int main()
{
      int i, pid;
      pid = fork();

      if (pid > 0)
      {
        sleep(2);
        return(0);
      }
      for (i=0; i<3; i++)
      {
        printf("My parent is %d\n", getppid());
        sleep(1);
      }

      return (0);
}
```

```
      #include <stdio.h>
      #include <stdlib.h>
      int main()
      {
            int i;
            int pid;
            pid = fork();
            for (i=1; i<=500; i++)
                if (pid > 0)
                    printf("%3i (parent)\n", i);
                 else
                    printf("%3i (child)\n", i);


      return (0);
```

Ερώτημα Β [20]:: Δημιουργήστε ένα πρόγραμμα στο οποίο μία διεργασία στο Linux/Unix παράγει άλλες 2 θυγατρικές της. Κάθε θυγατρική διαδικασία i (i=0,1), εκτελεί τον παρακάτω κώδικα, για να γράψει σε μία κοινά διαμοιραζόμενη μεταβλητή X με αρχική τιμή 0:

```
void child_process_i ( )
int tmp;        /* τοπικές μεταβλητές */
begin
for i = 1 to 500 do
{
        tmp = X;
        tmp = tmp+1;
        X = tmp;
}
end
```

Ποια ή ελάχιστη και ποια η μέγιστη τιμή της μεταβλητής X, για όλα τα δυνατά σενάρια σειριακής και παράλληλης εκτέλεσης? Εφαρμόστε τον αλγόριθμο του Peterson, ώστε να εξασφαλίσετε ότι η τελική τιμή του X θα πάρει τη μέγιστη τιμή της. Επίσης εξασφαλίστε ότι η πατρική διεργασία θα περιμένει την επιτυχή ολοκλήρωση των δύο θυγατρικών διεργασιών και μετά θα τερματίζει.

Μέρος 2 [70 μονάδες]

Ερώτημα Α [20]

Το παρακάτω πρόγραμμα εκτελείται ακολουθιακά και περιλαμβάνει έξι εντολές (E1, E2, … E6) εκχώρησης τιμής στις ακέραιες μεταβλητές A, B, C και D. Με την εκτέλεση των εντολών του προγράμματος υπολογίζονται οι τιμές των μεταβλητών A, B, C και D με βάση τις αρχικές τιμές των μεταβλητών X και Y (που θεωρούμε ότι είναι τιμές εισόδου για το πρόγραμμα δηλ. οποιεσδήποτε αρχικές ακέραιες τιμές). Κάθε εντολή εκχώρησης τιμής στο πρόγραμμα εκτελείται ατομικά – αδιαίρετα.

```
E1: A := X;
E2: B := Y;
E3: C := A + Y;
E4: D := C + B;
E5: C := D + B;
E6: D := D + A;
```

1. Να σχεδιάσετε ένα γράφο προτεραιοτήτων (precedence graph) που να αναπαριστά ισοδύναμα την εκτέλεση του παραπάνω ακολουθιακού προγράμματος με τη μέγιστη δυνατή παραλληλία **[5 μονάδες].**
2. Με χρήση εντολών cobegin … coend και εντολών begin … end (χωρίς να χρησιμοποιήσετε σημαφόρους και εντολές P (ή wait ή down) / V (ή signal ή up)) να δώσετε ένα πρόγραμμα που να είναι ισοδύναμο με το πιο πάνω ακολουθιακό πρόγραμμα με τη μέγιστη δυνατή παραλληλία **[7 μονάδες].**
3. Να δώσετε ένα «παράλληλο» πρόγραμμα, χρησιμοποιώντας την εντολή cobegin … coend και εντολές P (ή wait ή down) / V (ή signal ή up) χρησιμοποιώντας δυαδικούς σημαφόρους. Το παράλληλο πρόγραμμα που θα δώσετε πρέπει να υλοποιεί το γράφο προτεραιότητας που σχεδιάσατε στο Ερώτημα Α και να είναι ισοδύναμο με το πρόγραμμα που προτείνατε

στο Ερώτημα Β. Μην παραλείψετε να αρχικοποιήσετε τους δυαδικούς σημαφόρους που θα χρησιμοποιήσετε.

Σημείωση: Στο παράλληλο πρόγραμμα που θα δώσετε να προσπαθήσετε να χρησιμοποιήσετε το ελάχιστο απαραίτητο πλήθος δυαδικών σημαφόρων **[8 μονάδες].**

Ερώτημα Β [15 μονάδες]

Θεωρήστε τις διεργασίες Process1, Process2 και Process3 που εκτελούνται «παράλληλα» (δηλ. εκτελούνται σύμφωνα με το σχήμα εκτέλεσης "cobegin Process1; Process2; Process3; coend"). Κάθε διεργασία εκτελεί δύο αντίστοιχα τμήματα εντολών πριν να εκτελέσει ένα κώδικα κρίσιμου τμήματος. Για παράδειγμα, η διεργασία Process1 εκτελεί αρχικά το τμήμα εντολών Ε1.1, μετά εκτελεί το τμήμα εντολών Ε1.2 και τελικά εκτελεί τον κώδικα του κρίσιμου τμήματος.

Ο κώδικας των διεργασιών δίνεται ακολούθως:

Process1	**Process2**	**Process3**
E1.1;	E2.1;	E3.1;
E1.2;	E2.2;	E3.2;
<κρίσιμο τμήμα>;	<κρίσιμο τμήμα>;	<κρίσιμο τμήμα>;

Κατά την εκτέλεση των τριών διεργασιών πρέπει να εξασφαλιστούν οι ακόλουθες συνθήκες συγχρονισμού:

1. Το τμήμα εντολών Ε2.1 πρέπει να εκτελείται μετά από τα τμήματα εντολών Ε1.1 και Ε3.1.
2. Το τμήμα εντολών Ε1.2 πρέπει να εκτελείται μετά από το τμήμα εντολών Ε2.1.
3. Το τμήμα εντολών Ε.2.2 πρέπει να εκτελείται μετά από το τμήμα εντολών Ε.1.2.
4. Το τμήμα εντολών Ε3.2 πρέπει να εκτελείται μετά από το τμήμα εντολών Ε2.2.

Ζητείται να συμπληρώσετε τον παραπάνω κώδικα των τριών διεργασιών με εντολές P (ή wait ή down) και V (ή signal ή up) σε σημαφόρους (που πρέπει να αρχικοποιήσετε κατάλληλα), προκειμένου να εξασφαλίζονται όλες οι παραπάνω απαιτήσεις συγχρονισμού.

Ερώτημα Γ [15 μονάδες]

Παρακάτω παρουσιάζεται ο κώδικας δύο διεργασιών Producer και Consumer που εκτελούνται «παράλληλα» (δηλ. εκτελούνται με βάση το σχήμα "cobegin Producer; Consumer; coend") και υλοποιούν το γνωστό πρόβλημα συγχρονισμού του «Παραγωγού – Καταναλωτή».

Οι διεργασίες συγχρονίζονται για την προσπέλαση ενός κοινά διαμοιραζόμενου buffer 2 θέσεων που θεωρούμε ότι αρχικά είναι άδειος. Ο συγχρονισμός υλοποιείται με χρήση του δυαδικού σημαφόρου mutex και των δύο σημαφόρων μετρητών empty και full, που μπορούν να λαμβάνουν και αρνητικές τιμές.

```
binary semaphore mutex = 1;
counting semaphore  empty = 2;
counting semaphore  full = 0;
```

Διεργασία Producer	**Διεργασία Consumer**
while (TRUE) {	while (TRUE) {
<μη – κρίσιμο τμήμα>	<μη – κρίσιμο τμήμα>
wait(empty);	wait(full);
wait(mutex);	wait(mutex);

<εισαγωγή_στον_buffer>
signal(mutex);
signal(full);
<μη – κρίσιμο τμήμα>
}
	<εξαγωγή_από_τον_buffer>
	signal(mutex);
	signal(empty);
	<μη – κρίσιμο τμήμα>
	}

Στον ακόλουθο πίνακα παρουσιάζεται ένα συγκεκριμένο σενάριο εκτέλεσης (δυνατή ακολουθία εκτέλεσης) των εντολών wait και signal στους σημαφόρους μετρητές empty και full. Ζητείται να συμπληρώσετε τις δύο τελευταίες στήλες του πίνακα με τις τιμές που λαμβάνουν οι σημαφόροι empty και full μετά από την εκτέλεση κάθε αντίστοιχης εντολής signal ή wait που παρουσιάζεται στο συγκεκριμένο σενάριο.

Producer	Consumer	empty	full
		2	0
	wait(full)		
wait(empty)			
signal(full)			
wait(empty)			
	signal(empty)		
signal(full)			
wait(empty)			
signal(full)			
wait(empty)			
	wait(full)		
	signal(empty)		

<u>Ερώτημα Ε</u> [20]

Πέντε διεργασίες καταφθάνουν σε ένα υπολογιστικό σύστημα σύμφωνα με τα δεδομένα του ακόλουθου πίνακα:

Όνομα Διεργασίας	Χρονική Στιγμή Άφιξης	Απαιτήσεις Χρόνου Εκτέλεσης	Προτεραιότητα
P1	0	11	1
P2	2	4	4
P3	3	9	5
P4	3	12	3
P5	5	5	2

Σχεδιάζοντας τα κατάλληλα διαγράμματα Gantt, δείξτε πώς θα εκτελεστούν οι διεργασίες αυτές στην Κεντρική Μονάδα Επεξεργασίας (ΚΜΕ), και υπολογίστε τους μέσους χρόνους διεκπεραίωσης (ΜΧΔ) και αναμονής (ΜΧΑ), για κάθε έναν από τους παρακάτω αλγόριθμους χρονοδρομολόγησης:

- FCFS (First Come First Served). **[5]**
- SJF (Shortest Job First). **[5]**
- SRTF (Shortest Remaining Time First). **[5]**
- RR (Round Robin) με κβάντο χρόνου 2 χρονικές μονάδες. **[5]**

Παραδοχές:

1. Ο χρόνος εναλλαγής (context switch) είναι αμελητέος.
2. Για τον αλγόριθμο RR θεωρήστε πως αν τη χρονική στιγμή που μια νέα διεργασία έρχεται στο σύστημα διακόπτεται η εκτέλεση μιας διεργασίας (γιατί τελειώνει το κβάντο χρόνου της), τότε η νέα διεργασία εισέρχεται πριν από αυτήν που διακόπτεται στην ουρά των έτοιμων διεργασιών.

Καλή Επιτυχία!!!

Αριθμός μελών ανά ομάδα: *1 έως και 4.*
Ημερομηνία Παράδοσης: *Ημερομηνία Εξέτασης του μαθήματος στην εξεταστική Ιανουαρίου - Φεβρουαρίου*

Αποστολή εργασιών σε ZIP ή RAR μορφή (ο συμπιεσμένος φάκελος να περιέχει txt file με τα στοιχεία της ομάδας, pdf και C files) στο e-class καθώς και με e-mail στους διδάσκοντες:

makri@ceid.upatras.gr, aristeid@ceid.upatras.gr, sioutas@ceid.upatras.gr

Οι απαντήσεις που αφορούν κώδικα μέσα στο pdf της αναφοράς θα πρέπει να τεκμηριώνονται με screenshots από την εκτέλεση των προγραμμάτων.