Λειτουργικά Συστήματα

2021 - 2022

1η Εργαστηριακή Άσκηση

Παύλος Πεσκελίδης, ΑΜ: 1072483

Βασιλική – Ευαγγελία Δούρου, ΑΜ: 1072633

Δημήτριος Μπαλάφας, ΑΜ: 1072499

Έτος 3ο, Εξάμηνο 5ο

Μέρος 1ο

Ερώτημα Α:

Κώδικας 1ος

Οι πρώτες δύο γραμμές εισάγουν header αρχεία.
Στη main(), ορίζουμε δύο μεταβλητές εκ των οποίων η μία είναι μετρητής. Έπειτα, καλώντας την συνάρτηση fork(), δημιουργούμε μία διεργασία παιδί πανομοιότυπη με την διεργασία γονέα, ενώ και οι δύο διεργασίες θα εκτελέσουν ταυτόχρονα τον παρακάτω κώδικα.
Σημείωση : Η fork() επιστρέφει θετική τιμή στη διεργασία γονέα, η οποία είναι το pid (id της διεργασίας) της διεργασίας παιδιού που μόλις δημιουργήθηκε, ενώ επιστρέφει 0 στη διεργασία παιδί.
Επομένως, η if θα εκτελεστεί μόνο από τη διεργασία γονέα. Η διεργασία που θα εκτελέσει την if, θα περιμένει για 2 δευτερόλεπτα, λόγω της sleep(2) και έπειτα θα τερματιστεί. Η διεργασία παιδί θα εκτελέσει τη for, όπου κάθε φορά θα επιστρέφει το pid της διεργασίας γονέα και θα αναμένει για ένα δευτερόλεπτο σε κάθε επανάληψη (sleep(1)). Όπως βλέπουμε στο παρακάτω στιγμιότυπο, από την εκτέλεση του προγράμματος στις 2 πρώτες επαναλήψεις θα επιστρέφεται το pid της διεργασίας γονέα, ενώ στη τελευταία επανάληψη (i=2) θα επιστρέφεται η τιμή 1, η οποία είναι το pid της διεργασίας init. Αυτό συμβαίνει, καθώς η διεργασία γονέας θα έχει τερματίσει μετά από 2 δευτερόλεπτα αναμονής(από την if) και έτσι η διεργασία παιδί θα ανατεθεί εκ νέου ως παιδί της διεργασίας init.

Τρέχοντας τον κώδικα εμφανίζεται το εξής αποτέλεσμα:

```
My parent is 7903
My parent is 7903
My parent is 1
```

Κώδικας 2ος

Οι πρώτες δύο γραμμές εισάγουν header αρχεία.
Στη main(), όπως και στον πρώτο κώδικα ορίζουμε δύο μεταβλητές εκ των οποίων η μία είναι μετρητής. Έπειτα, καλώντας την συνάρτηση fork(), δημιουργούμε μία διεργασία παιδί πανομοιότυπη με την διεργασία γονέα, ενώ και οι δύο διεργασίες θα εκτελέσουν ταυτόχρονα τον παρακάτω κώδικα.
Σημείωση : Η fork() επιστρέφει θετική τιμή στη διεργασία γονέα, η οποία είναι το pid (id της διεργασίας) της διεργασίας παιδιού που μόλις δημιουργήθηκε, ενώ επιστρέφει 0 στη διεργασία παιδί.
Η for() θα εκτελεστεί και από τη διεργασία γονέα και από τη διεργασία παιδί. Ωστόσο, η διεργασία γονέας θα εκτυπώνει την πρώτη printf() αφού η τιμή που της επιστρέφει η fork() είναι θετική(και έτσι περνάει στην if), ενώ η διεργασία παιδί θα εκτελέσει το δεύτερο printf(), καθώς η τιμή που της επιστρέφει η fork() είναι μηδέν και έτσι θα εκτελέσει το else.

Τρέχοντας τον κώδικα εμφανίζεται το εξής απόσπασμα του αποτελέσματος:

```
260 (child)
293 (parent)
261 (child)
294 (parent)
262 (child)
295 (parent)
263 (child)
296 (parent)
264 (child)
297 (parent)
265 (child)
298 (parent)
266 (child)
299 (parent)
267 (child)
300 (parent)
268 (child)
301 (parent)
269 (child)
302 (parent)
303 (parent)
304 (parent)
305 (parent)
306 (parent)
307 (parent)
308 (parent)
309 (parent)
310 (parent)
311 (parent)
312 (parent)
313 (parent)
314 (parent)
315 (parent)
316 (parent)
317 (parent)
318 (parent)
319 (parent)
320 (parent)
321 (parent)
322 (parent)
323 (parent)
324 (parent)
325 (parent)
326 (parent)
327 (parent)
328 (parent)
329 (parent)
330 (parent)
331 (parent)
332 (parent)
```

Ερώτημα Β:

Για τη σειριακή εκτέλεση έχουμε δημιουργήσει τη δοσμένη συνάρτηση child_process_i και στη main() έχουμε δημιουργήσει μια διαμοιραζόμενη μεταβλητή X που αρχικοποιείται με 0 και δύο child processes, εκτελώντας τη συνάρτηση fork(). Όταν η τιμή που επιστρέφεται στις δύο μεταβλητές i_1 , i_2 από τη fork() είναι ίση με το μηδέν, τότε καλούμε τη child_process_i και από τα

δύο παιδία. Τέλος, εκτυπώνεται η τιμή της μεταβλητής X. Ο κώδικας για τη σειριακή εκτέλεση, είναι στο zip.

Το αποτέλεσμα που προκύπτει από την εκτέλεση του κώδικα είναι το παρακάτω:

```
1000
```

Σε αυτή τη περίπτωση, για τη παράλληλη εκτέλεση έχουμε πάλι μία συνάρτηση child_process_i() και στη main() αρχικοποιούμε μία διαμοιραζόμενη μεταβλητή X = 0 και παράγουμε δύο θυγατρικές διεργασίες, μέσω της fork(). Αν το αποτέλεσμα της fork() είναι μηδέν και για τις δύο (αφού η fork επιστρέφει μηδέν στις παραγόμενες διεργασίες), τότε εκτελούν και οι δύο τη συνάρτηση child_process_i. Τέλος, το αποτέλεσμα εκτυπώνεται από το γονέα της P_1.

Ο κώδικας για τη παράλληλη εκτέλεση είναι στο zip.

Το αποτέλεσμα που προκύπτει από την εκτέλεση του κώδικα είναι το παρακάτω:

```
X=500
X=500
```

Λαμβάνοντας υπόψη τον αλγόριθμο του Peterson προκύπτει ο κώδικας που επισυνάπτεται στο zip. Σε αυτή τη περίπτωση, έχουμε δημιουργήσει τρεις συναρτήσεις. Η πρώτη είναι η enter_region(), η οποία καλείται, έτσι ώστε μία διεργασία να μπει σε κρίσιμη περιοχή. Η δεύτερη είναι η leave_region(), η οποία καλείται για να βγει μία διεργασία από τη κρίσιμη περιοχή και η τρίτη είναι η child_process_i(), η οποία περιέχει την κρίσιμη περιοχή. Παράλληλα, έχουμε έναν διαμοιραζόμενο πίνακα interested[2], στον οποίο θέτουμε 1, αν μία διεργασία ενδιαφέρεται να μπει στη κρίσιμη περιοχή της και 0 αν όχι και μία διαμοιραζόμενη μεταβλητή turn που χρησιμοποιείται για να δώσει σειρά σε μία διεργασία. Στη main() αρχικοποιείται τυχαία η μεταβλητή turn με τιμή 1, ώστε να δώσουμε προτεραιότητα στη δεύτερη θυγατρική διεργασία και δημιουργούνται δύο θυγατρικές διεργασίες, οι οποίες καλούν τη συνάρτηση child_process_i() και εισάγονται εναλλάξ στη κρίσιμη περιοχή τους. Επιπρόσθετα, έχουμε χρησιμοποιήσει τη συνάρτηση waitpid() για να περιμένει η γονική διεργασία να τελειώσουν τις εκτελέσεις τους οι θυγατρικές διεργασίες.

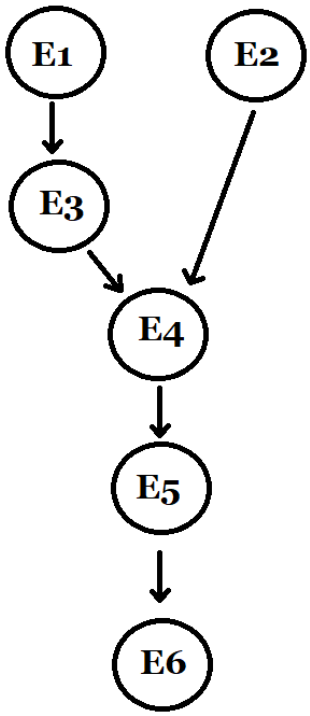
Το αποτέλεσμα που προκύπτει από την εκτέλεση του κώδικα είναι το παρακάτω:

```
I'm 10401, my father is 10093 and for me X is 500
I'm 10403, my father is 10093 and for me X is 1000
```

Μέρος 2ο

Ερώτημα Α :

1. Ο ζητούμενος γράφος προτεραιοτήτων του δοσμένου ακολουθιακού προγράμματος είναι ο παρακάτω :



2. Το ζητούμενο πρόγραμμα σε ψευδογλώσσα είναι το ακόλουθο:

```
cobegin

        E2;

        begin
        E1;
        E3;
        end;

coend;

        E4;
        E5;
        E6;
```

3. Το ζητούμενο πρόγραμμα με τη χρήση δυαδικών σηματοφόρων είναι το ακόλουθο, όπου με Σ_{ij} είναι οι σηματοφόροι για τον έλεγχο των προτεραιοτήτων E_i -> E_j, οι οποίοι αρχικοποιούνται με τη

τιμή 0, εκτός του Σ4, ο οποίος αρχικοποιείται με -1 και ελέγχει την εξάρτηση ανάμεσα στις E3 και E2 με την E4.

```
var Σ13,Σ4,Σ45,Σ56 : semaphores;
Σ13 := Σ45 := Σ56 := 0;
Σ4 := -1;
cobegin

        begin E1; up(Σ13); end;
        begin E2; up(Σ4); end;
        begin down(Σ13); E3; up(Σ4); end;
        begin down(Σ4); E4; up(Σ45); end;
        begin down(Σ45); E5; up(Σ56); end;
        begin down(Σ56); E6; end;

coend
```

Ερώτημα Β:

Ο ζητούμενος κώδικας είναι ο ακόλουθος, όπου με Σ_{ij} είναι οι σημαφόροι για τον έλεγχο των προτεραιοτήτων E_i -> E_j, οι οποίοι αρχικοποιούνται με τη τιμή 0. Παράλληλα, αρχικοποιείται ένας σημαφόρος mutex με τιμή 1, ο οποίος χρησιμοποιείται για να μπαίνει και να βγαίνει μία διεργασία από τη κρίσιμη περιοχή της.

```
var Σ1121,Σ3121,Σ2112,Σ1222,Σ2232,mutex;

Σ1121:=Σ3121:=Σ2112:=Σ1222:=Σ2232:=0;
mutex:=1;

cobegin

//Process 1             //Process 2             //Process 3
begin                   begin                   begin
E1.1;                   down(Σ3121)             E3.1;
up(Σ1121);              down(Σ1121);            up(Σ3121);
down(Σ2112);            E2.1;                   down(Σ2232);
E1.2;                   up(Σ2112);              E3.2;
up(Σ1222);              down(Σ1222);            down(mutex);
down(mutex);            E2.2;                   <critical region>;
<critical region>;      up(Σ2232);              up(mutex);
up(mutex);              down(mutex);            end;
end;                    <critical region>;
                        up(mutex);
                        end;
coend
```

Ερώτημα Γ:

Ο συμπληρωμένος πίνακας που ζητείται είναι ο ακόλουθος :

Producer	Consumer	empty	full
		2	0
	wait(full)	2	-1
wait(empty)		1	-1
signal(full)		1	0
wait(empty)		0	0
	signal(empty)	1	0
signal(full)		1	1
wait(empty)		0	1
signal(full)		0	2
wait(empty)		-1	2
	wait(full)	-1	1
	signal(empty)	0	1

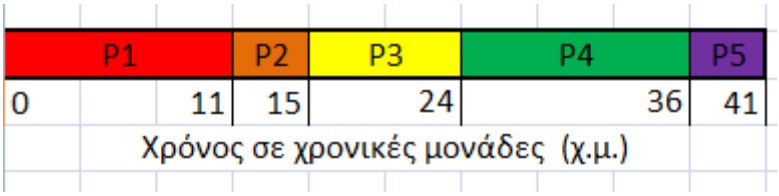
Ερώτημα Δ:

Σημείωση: Στην εκφώνηση του project ήταν γραμμένο ως Ε και δεν υπήρχε ερώτημα Δ.

Παραδοχές : Θεωρούμε ότι μεγάλοι αριθμοί σηματοδοτούν μεγάλες προτεραιότητες, δηλαδή ότι η μέγιστη προτεραιότητα είναι 5 και η ελάχιστη 1. Σε περίπτωση, που δύο διεργασίες έχουν τον ίδιο χρόνο αύξησης, πρώτη θα εκτελεστεί αυτή με τη μεγαλύτερη προτεραιότητα.

1. FCFS

Το διάγραμμα Gantt είναι το ακόλουθο.



Εάν είναι t_1 η χρονική στιγμή εισόδου και t_2 η χρονική στιγμή εξόδου, τότε ο χρόνος διεκπεραίωσης είναι $ΧΔ = t_2 - t_1$ και για κάθε διεργασία είναι ίσος με :

$ΧΔ_{P1} = 11 - 0 = 11$, $ΧΔ_{P2} = 15 - 2 = 13$, $ΧΔ_{P3} = 24 - 3 = 21$, $ΧΔ_{P4} = 36 - 3 = 33$, $ΧΔ_{P5} = 41 - 5 = 36$

Άρα, ο μέσος χρόνος διεκπεραίωσης θα είναι :

$ΜΧΔ = 114 / 5 = 22,8$.

Ο χρόνος αναμονής είναι ίσος με το χρόνο διεκπεραίωσης μείον το χρόνο που χρειάζεται η διεργασία για εξυπηρέτηση από την ΚΜΕ, δηλαδή είναι ΧΑ = ΧΔ – t_{CPU} και για κάθε διεργασία είναι ίσος με :

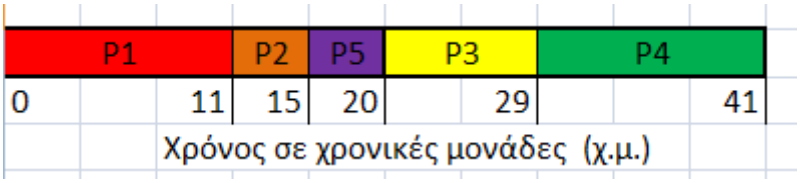
$ΧΑ_{P1}$ = 11 - 11 = 0 , $ΧΑ_{P2}$ = 13 - 4 = 9, $ΧΑ_{P3}$ = 21 - 9 = 12, $ΧΑ_{P4}$ = 33 - 12 = 21, $ΧΑ_{P5}$ = 36 - 5 = 31

Άρα, ο μέσος χρόνος αναμονής θα είναι :

ΜΧΑ = 73 / 5 = 14,6.

2.SJF

Το διάγραμμα Gantt είναι το ακόλουθο.



Εάν είναι t_1 η χρονική στιγμή εισόδου και t_2 η χρονική στιγμή εξόδου, τότε ο χρόνος διεκπεραίωσης είναι ΧΔ = t_2 – t_1 και για κάθε διεργασία είναι ίσος με :

$ΧΔ_{P1}$ = 11 - 0 = 11 , $ΧΔ_{P2}$ = 15 - 2 = 13, $ΧΔ_{P3}$ = 29 - 3 = 26, $ΧΔ_{P4}$ = 41 - 3 = 38, $ΧΔ_{P5}$ = 20 - 5 = 15

Άρα, ο μέσος χρόνος διεκπεραίωσης θα είναι :

ΜΧΔ = 103 / 5 = 20,6.

Ο χρόνος αναμονής είναι ίσος με το χρόνο διεκπεραίωσης μείον το χρόνο που χρειάζεται η διεργασία για εξυπηρέτηση από την ΚΜΕ, δηλαδή είναι ΧΑ = ΧΔ – t_{CPU} και για κάθε διεργασία είναι ίσος με :

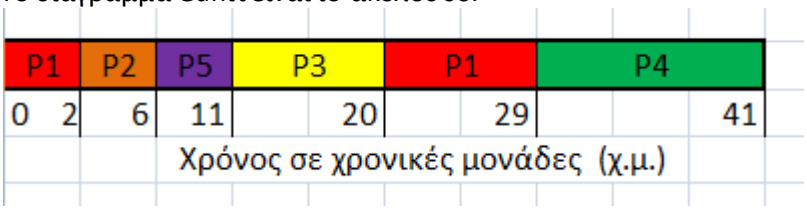
$ΧΑ_{P1}$ = 11 - 11 = 0 , $ΧΑ_{P2}$ = 13 - 4 = 9, $ΧΑ_{P3}$ = 26 - 9 = 17, $ΧΑ_{P4}$ = 38 - 12 = 26, $ΧΑ_{P5}$ = 15 - 5 = 10

Άρα, ο μέσος χρόνος αναμονής θα είναι :

ΜΧΑ = 62 / 5 = 12,4.

3.SRTF

Το διάγραμμα Gantt είναι το ακόλουθο.



Εάν είναι t_1 η χρονική στιγμή εισόδου και t_2 η χρονική στιγμή εξόδου, τότε ο χρόνος διεκπεραίωσης είναι ΧΔ = t_2 – t_1 και για κάθε διεργασία είναι ίσος με :

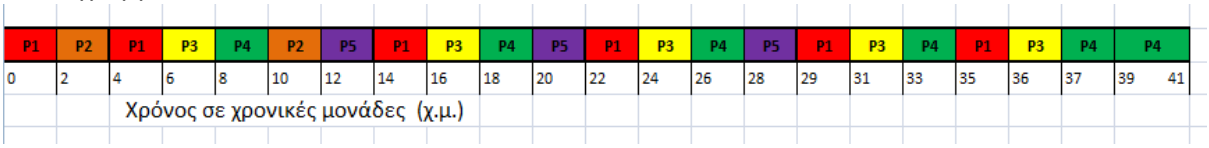
$ΧΔ_{P1}$ = 29 - 0 = 29 , $ΧΔ_{P2}$ = 6 - 2 = 4, $ΧΔ_{P3}$ = 20 - 3 = 17, $ΧΔ_{P4}$ = 41 - 3 = 38, $ΧΔ_{P5}$ = 11 - 5 = 6

Άρα, ο μέσος χρόνος διεκπεραίωσης θα είναι :

ΜΧΔ = 94 / 5 = 18,8.

Ο χρόνος αναμονής είναι ίσος με το χρόνο διεκπεραίωσης μείον το χρόνο που χρειάζεται η διεργασία για εξυπηρέτηση από την ΚΜΕ, δηλαδή είναι ΧΑ = ΧΔ – t_{CPU} και για κάθε διεργασία είναι ίσος με :
$ΧΑ_{P1}$ = 29 - 11 = 18 , $ΧΑ_{P2}$ = 4 - 4 = 0, $ΧΑ_{P3}$ = 17 - 9 = 8, $ΧΑ_{P4}$ = 38 - 12 = 26, $ΧΑ_{P5}$ = 6 - 5 = 1
Άρα, ο μέσος χρόνος αναμονής θα είναι :
ΜΧΑ = 53 / 5 = 10,6.

4.RR
Το διάγραμμα Gantt είναι το ακόλουθο.



Εάν είναι t_1 η χρονική στιγμή εισόδου και t_2 η χρονική στιγμή εξόδου, τότε ο χρόνος διεκπεραίωσης είναι ΧΔ = t_2 – t_1 και για κάθε διεργασία είναι ίσος με :
$ΧΔ_{P1}$ = 36 - 0 = 36 , $ΧΔ_{P2}$ = 12 - 2 = 10, $ΧΔ_{P3}$ = 37 - 3 = 34, $ΧΔ_{P4}$ = 41 - 3 = 38, $ΧΔ_{P5}$ = 29 - 5 = 24
Άρα, ο μέσος χρόνος διεκπεραίωσης θα είναι :
ΜΧΔ = 142 / 5 = 28,4.

Ο χρόνος αναμονής είναι ίσος με το χρόνο διεκπεραίωσης μείον το χρόνο που χρειάζεται η διεργασία για εξυπηρέτηση από την ΚΜΕ, δηλαδή είναι ΧΑ = ΧΔ – t_{CPU} και για κάθε διεργασία είναι ίσος με :
$ΧΑ_{P1}$ = 36 - 11 = 25 , $ΧΑ_{P2}$ = 10 - 4 = 6, $ΧΑ_{P3}$ = 34 - 9 = 25, $ΧΑ_{P4}$ = 38 - 12 = 26, $ΧΑ_{P5}$ = 24 - 5 = 19
Άρα, ο μέσος χρόνος αναμονής θα είναι :
ΜΧΑ = 101 / 5 = 20,2.