Page Faults

1) Βέλτιστη αντικατάσταση
2) Fifo
3) LRU

Άσκηση 3.1

Μνήμη 128 MB εκχωρείται σε μονάδες των n bytes.
Πόσος αποθηκευτικός σε bytes χρειάζεται για την παρακολούθηση της ελεύθερης μνήμης
Α) με χάρτη bit (bitmap) } Ποια μέθοδος
Β) με συνδεδεμένη λίστα } είναι καλύτερη

Για την συνδεδεμένη λίστα υποθέστε ότι:
i) Η μνήμη αποτελείται από μία εναλλασσόμενη ακολουθία τμημάτων δεδομένων και Κενών (οπών) σε μέγεθος 64 KB
ii) Κάθε κόμβος χρειάζεται μία διεύθυνση μνήμης των 32bit ένα πεδίο 16 bit για το μήκος, και ένα πεδίο 16 bit για τον επόμενο κόμβο.

Λύση:

1 KB = 2^{10} Byte και 1 MB = 2^{20} byte
Άρα η μνήμη 128 MB = $2^7 * 2^{20}$ byte = 2^{27} byte

α) Ο bitmap χρειάζεται 1 bit για κάθε μονάδα κατανομής
Άρα για 2^{27} byte μνήμης, ο χάρτης χρειάζεται χώρο $\frac{2^{27}}{n}$ bit

β) Η συνδεδεμένη λίστα χρειάζεται (32+16+16) = 64 bit/κόμβο
= 8 byte = 2^3 byte / κόμβο

Εφόσον κάθε τμήμα έχει μήκος 64KB (=2^6*2^{10}byte=2^{16} byte), συνολικά θα υπάρχαν $2^{27}/2^{16}=2^{11}$ κόμβοι σκληρού. Άρα συνολικός χώρο 2^{11} * 2^3 byte/κόμβο = 2^{14} byte

κόμβοι

Συνεπώς, εαν κάθε μονάδα έχει μήκος $\eta > 2^{13}$ byte, ο bitmap είναι καλύτερος.

Ασκ 3.2 ;?

Ασκηση 3.3

Μια μηχανή έχει χώρο διευθύνσεων 32 bit και σελίδες μεγέθους 8 KB. Ο πίνακας σελίδων βρίσκεται εξολοκλήρου στο υλικό και χρησιμοποιεί μία λέξη των 32bit για κάθε καταχώρηση. Όταν ξεκινάει μία διεργασία, ο πίνακας σελίδων αντιγράφεται από την μνήμη στο υλικό, με ρυθμό μία λέξη ανά 100 nsec.

Αν κάθε διεργασία

Χώρος διευθύνσεων N $\overset{\text{Άρα}}{\Rightarrow}$ 2^N διευθύνσεις

Ασκηση 3.7

45

Το linux χρησιμοποιεί 12 bit για ασφάλεια

Η φυσική μνήμη είναι κάτι υπαρκτό ενώ η εικονική μνήμη Βρίσκεται μόνο στο χώρο των προγραμμάτων - μη υπαρκτός χώρος διευθύνσεων

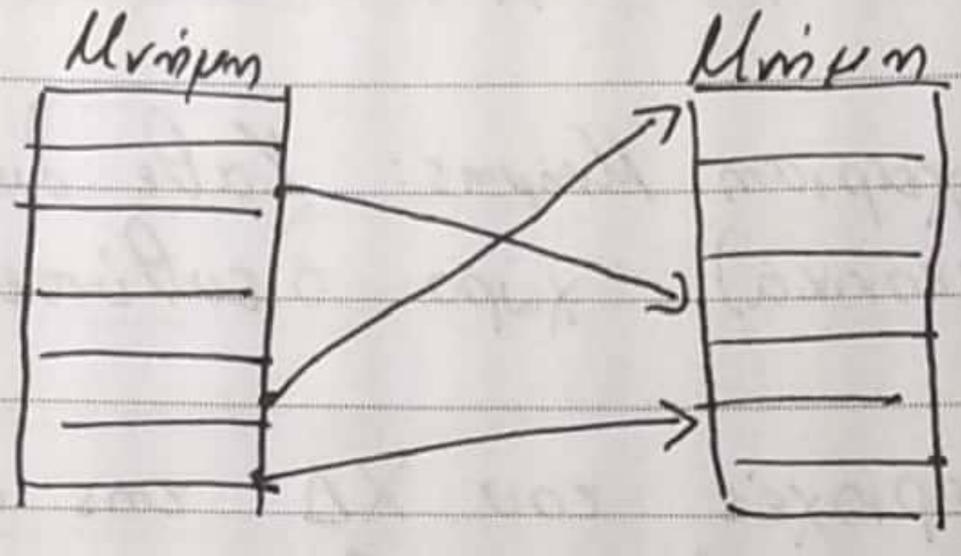
Διαχείριση Μνήμης: Κάθε διεργασία έχει το δικό της (εικονικά) χώρο διευθύνσεων (ΧΔ)

Περιοχές του ΧΔ της διεργασίας
- Αντιστοιχίζονται σε περιοχές της φυσικής μνήμης
- Αποθηκεύονται (προσωρινά) σε δευτερεύουσα συσκευή αποθήκευσης

Εικονική Μνήμη: Οι περισσότερες διεργασίες χρησιμοποιούν μόνο μικρό μέρος του εικονικού χώρου Διευθύνσεων (ΧΔ) τους.

Ζήτημα 2 (1 Μονάδα)

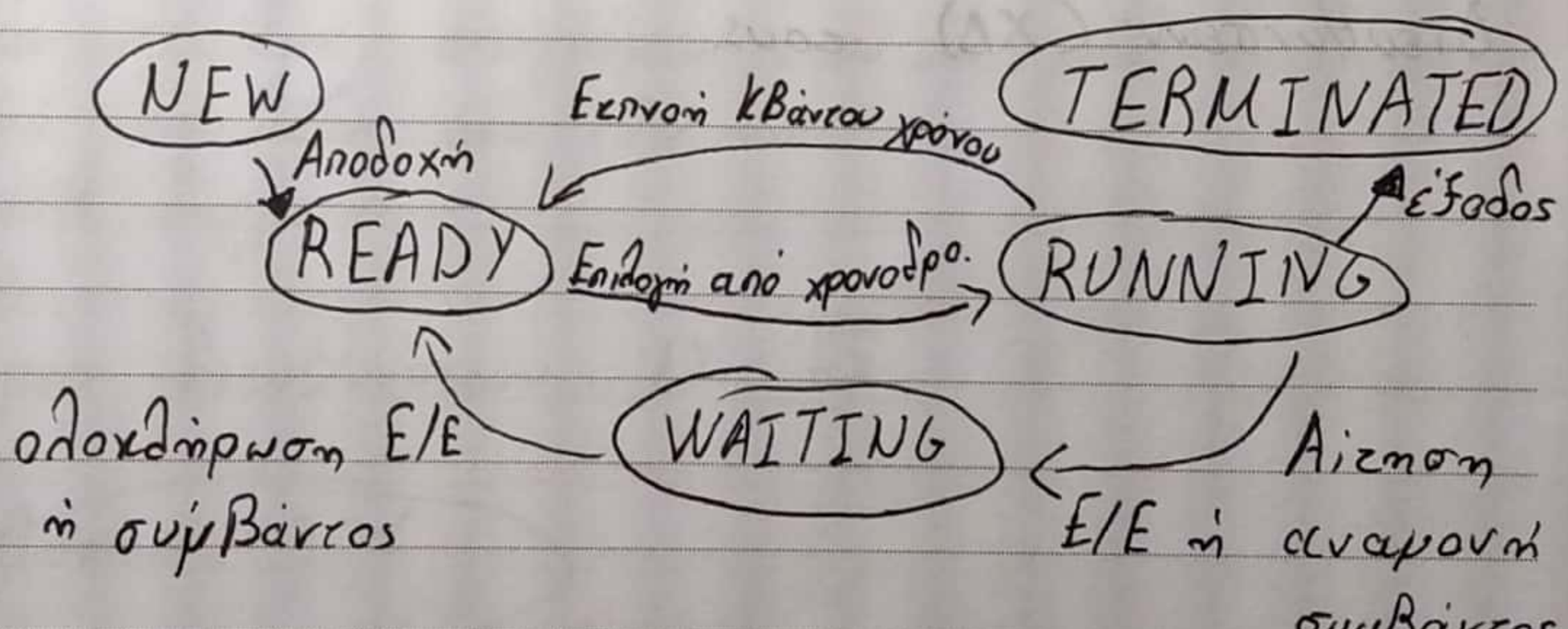
Η ΜΜΟ χαρτογραφεί τις εικονικές διευθύνσεις και τις αντιστοιχεί σε διευθύνσεις της φυσικής μνήμης (δηλαδή της RAM). Π.χ. Εικονική Φυσική



Ζήτημα 1 [3 Μονάδες]

α) Σχεδιάστε ενδεικτικό διάγραμμα καταστάσεων διεργασίας με τις καταστάσεις NEW, RUNNING, WAITING, READY, TERMINATED

Λύση:



c) → Αίτηση διακοπής από λειτουργικό (trap, software interrupt)
→ Ολοκλήρωση λειτουργίας Ε/Ε (διακοπή υλικού, hardware interrupt)
→ Λάθος ή με επιτρεπτή εντολή: π.χ διαίρεση με το 0 (exception, εξαίρεση)

Ζήτημα 4 2 Μονάδες

Δύο διεργασίες A, B φτάνουν για εκτέλεση τη χρονική στιγμή 0 σε ένα CPU που υποστηρίζει round robin scheduling με κβάντο χρόνου 5μsec. Οι διεργασίες έχουν χρόνους εκτέλεσης 30μsec και 60 μsec αντίστοιχα. Κάθε εναλλαγή διεργασίας στη CPU προσθέτει επιβάρυνση 5μsec. Δείξτε αναλυτικά τα βήματα εκτέλεσης των A, B διεργασιών σε σχέση με το χρόνο. Ποια χρονική στιγμή θα ολοκληρώσει την εκτέλεσή της η κάθε διεργασία;

Χρόνος	Ουρά	ΚΜΕ
0	A(A,30μsec) B(60μsec)	A
4	(A,25) (B,60) (B,60)	(A,25) -
29	(B,60) (A,25)	B
34	(A,25) (B,55)	-
39	(A,25) (B,55)	A
64	(B,55) (A,20)	-
69		

Ζήτημα 2 3.5 μονάδες

Πως δημιουργείται deadlock; Δώστε παράδειγμα με 2 διαδικασίες που εμπλέκονται σε deadlock όταν εκτελούν αντίστοιχα read και write. Ποια η διαφορά του με starvation;

Deadlock δημιουργείται όταν καθε μέλος μιάς ομάδας περιμένει κάποιο άλλο μέλος να αναλάβει δράση όπως η αποστολή ενός μηνύματος ή η απελευθέρωση μιας κλειδαριάς

π.χ.

Η process 1() θα διαβάσει δεδομένα από το cd και θα τα αποθηκεύσει σε μαγνητική ταινία.

Η process 2() θα διαβάζει από την μαγνητική ταινία και θα τα αποθηκεύει στο cd.

Η διαφορά με starvation είναι ότι σε αυτή την περίπτωση σε μία διαδικασία, απαγορεύονται συνεχώς οι απαραίτητοι όροι για να επεξεργαστεί το έργο της.

Ζήτημα 4 [1]

Χώρος λογικών διευθύνσεων 256 σελίδων με 1024 λέξεις η κάθε μία, ο οποίος αντιστοιχίζεται σε μία φυσική μνήμη 16 πλαισίων. Πόσα bits υπάρχουν μέσα στη λογική διεύθυνση? Πόσα bits υπάρχουν μέσα στη φυσική διεύθυνση;

Λύση: $256 \to 2^{8}$ άρα 8 bits για την κωδικοποίηση 256 αριθμών σελίδων

$1024 \to 2^{10}$ άρα 10 bits για την κωδικοποίηση μετατόπισης σε μία σελίδα 1024 λέξεων

Άρα 8+10 = 18 bits υπάρχουν μέσα στη λογική διεύθυνση.

16 πλαίσια $\to 2^{4}=16$ άρα 4 bits για την κωδικοποίηση 16 αριθμών πλαισίων

$1024 \to 2^{10}=1024$ άρα 10 bits για την κωδικοποίηση σε ένα πλαίσιο 1024 λέξεων.

άρα 4+10 = 14 bits υπάρχουν μέσα στη φυσική διεύθυνση.

Page Faults

Βέλτιστη Αντικατάσταση

Μια διεργασία εκτελεί την ακολουθία αναφορών σελιδών: 1, 3, 4, 3, 6, 1, 3, 4, 3. Θεωρώντας ότι 3 πλαίσια μνήμης είναι αρχικά άδεια, βρείτε τον αριθμό σφαλμάτων

Αναφορά	Διαθέσιμα πλαίσια μνήμης			Σφάλμα
1	1			ΝΑΙ
3	3	1		ΝΑΙ
4	4	3	1	ΝΑΙ
3	4	3	1	
6	6	3	1	ΝΑΙ
1	6	3	1	
3	6	3	1	
4	4	3	1	ΝΑΙ
3	4	3	1	

5 σφάλματα

Αν δεν υπάρχει η αναφορά σε κάποιο πλαίσιο μνήμης τοτε βάζω στο 1ο πλαίσιο.

FiFo

Αναφορά	πλαίσια Μνήμης			Σφαλμα
1	1			ΝΑΙ
3	3	1		ΝΑΙ
4	4	3	1	ΝΑΙ
3	4	3	1	
6	6	4	3	ΝΑΙ
1	1	6	4	ΝΑΙ
3	3	1	6	ΝΑΙ
4	4	3	1	ΝΑΙ
3	4	3	1	

7 Σφάλματα

Αν δεν υπάρχει η αναφορά μεταφέρω τα πλαίσια ένα δεξιά και βάζω στο 1ο την νέα αναφορα.

LRU

Αναφορά	Πλαίσια Μνήμης			Σφάλμα
1	1			ΝΑΙ
3	3	1		ΝΑΙ Αν υπάρχει η αναφορά
4	4	3	1	ΝΑΙ μπαίνει στο πρώτο πλαίσιο
3	3	4	1	και τα υπόλοιπα πλαίσια
6	6	3	4	ΝΑΙ πάνε δεξιά. Αν δεν υπάρχει
1	1	6	3	ΝΑΙ μπαίνει μπροστά η καινούργια
3	3	1	6	και οι παλιές 1 πλαίσιο
4	4	3	1	ΝΑΙ δεξιά.
3	3	4	1	

6 Σφάλματα

Άσκηση 3.2

Κενές οπές: 10KB, 4KB, 20KB, 18KB, 7KB, 9KB, 12KB, 15KB
Ποια οπή θα χρησιμοποιηθεί αν γίνουν συνεχόμενες αιτήσεις τμημάτων με μέγεθος Ⓐ 12KB Ⓑ 10KB Ⓒ 9KB
και χρησιμοποιείται ο αλγόριθμος της πρώτης προσαρμογής;

Πρώτη προσαρμογή: 20KB, 10KB, 18KB
Πάω στις κενές οπές και βλέπω που χωράνε τα τμήματα μου. π.χ. για το Ⓐ βλέπω ξεκινάω από αριστερά στις κενές οπές και βλέπω ότι η 1η που χωράει είναι η 20KB. Έπειτα συνεχίζω για τις άλλες

Βέλτιστη προσαρμογή: 12KB, 10KB, 9KB

Επόμενη προσαρμογή: 18KB, 20KB, 9KB

Χείριστη προσαρμογή: 20KB, 18KB, 15KB

Λειτουργικά Συστήματα Ι

Fork() sleep() exit()
n = Fork()
iF n<-1 error
iF n=0 child
iF n>0 parent

getpid(), getppid() <unistd.h>
pid-t pid;
pid = Fork() ή pid = getppid();

wait()
<sys/wait.h>
until child over

pthreads | <pthread.h>, <stint_t.h>
pthread_t tid[3], int i; (Δικλωση)
For (i=0; i<3; i++)
pthread_create (&tid[i], NULL, συναρτησιμη);

pthread_yield() → επιτρέπει σε ένα νήμα να εγκαταλείψει τον επεξεργαστή ώστε ένα άλλο νήμα να έχει την ευκαιρία να τρέξει
For ....
pthread_join [tid[i], NULL); → Κάνει block μέχρι το νήμα με id thread να τερματίσει.

MUTEX:
pthread_mutex_t L = PTHREAD_MUTEX_INITIALIZER;
-lock (&L);, pthread_mutex_unlock (&L);
-trylock (&L);

Κοινη μνημη:
int shmid, *mygvars | shmid = shmget(IPC_PRIVATE, 4, IPC_CREAT\0666);
πρεπει να εδεξουν → mygvars = shmat (shmid, 0, 0);
shmctl (shmid, IPC_RMID, 0);

Semaphore: #include <semaphore.h>
sem_t occ; δηλωση
sem_init (&occ, 0, τιμη); αρχικοποιηση
sem_wait(&occ); ελαττωνει κατα 1 οσο
sem_post (&occ); Αν δεν κατα 1 την τιμη του semaphore.

2 παραδειγμα 2 thread
Void *prosthesi 1(); | #include <pthread.h>
- ,,- 2(); | -,,- <stint.h>
sem_t semA; | pthread_create(&tid[0], NULL,
int x=100, y=0; | -,,- a+b);
sem_init (&semA, 0, 0); | pthread_join [tid[0], NULL);

Posix shared memory: (shm_open, &truncate, mmap):
#define MUTEX "/lock"
int segment_mutex, mode = S_IRWXU | S_IRWXG;
segment_mutex = shm_open(MUTEX, O_CREAT | O_RDWR, 0, TRUNC, mode);
Ftruncate(segment_mutex, sizeof(pthread_mutex_t));
Pthread_mutex_t * BO;
BO = (pthread_mutex_t *)mmap(Null, sizeof(pthread_mutex_t), PROT_READ | PROT_WRITE, MAP_SHARED, segment_mutex, -1);

Barrier pthread:

pthread_barrier_t bar1; ορισμος
pthread_barrier_init(&bar1, NULL, συν); Αρχικοποιηση
pthread_barrier_wait(&bar1); Κλειδωνει εως οτου ολα τα thread παν εκει

Pipes

int P[2] P[0]=read P[1]=write
If (pipe(p)<0): εχω fail exit(1)
read (p[0], Μεταβλητη απο που θελουμε, μεγεθος συμβολοσειρας);
write (P[1], -''- , -''-);
ο ότι γραφεται στο P[1] διαβαζεται απο το P[0]
includes: <sys/types>.h, stdlib.h, stdio.h> unistd.h, sys/ipc.h, sys/shm.h, stdint.h, pthread.h.

```
#include <sys/types.h>
#include <stdio.h>

int main(void){
int mypipe[2];
int mypipe2[2];
int mypipe3[2];
int mypipe4[2];
int A=5,i,n,n2,n3,n4//n=parent n2=
pipe(mypipe); ] n2==0,n3,n4 παιδια
pipe(mypipe2);
                n2>0
        };
    };
n=Fork();
 if (n==-1)
  FprintF(stream,problem creating process\n");
 elseiF (n==0)
  n2=Fork(); //δημιουργειται γκαμπινας process
 iF (n2==0)
  close(mypipe2[1]);
```

```
read(mypipe2[0],&A,sizeof(A));
printF("egkoni (%d) received value %d\n",getpid(),A);
 close(mypipe2[0]);
}else iF (n2>0){
A+=1;
write(mypipe2[1],&A,sizeof(A));
printf("mpampas (%d) send value: %d\n",getpid(),A);
 close(mypipe3[1]);
 read(mypipe3[0],&A,sizeof(A));
printF("egkoni 2 (%d) received value: %d\n",getpid(),A);
close(mypipe3[0]);
else iF (n3>0)
close(mypipe3[0]);
A+=2;
write(mypipe3[1],&A,sizeof(A));
```

```
m4=Fork(); iF(m4==0){
close(mypipe4[1]);
read(mypipe4[0],&A,sizeof(A));
printF(egkoni 3 (%d) received value %d\n",getpid(),A);
close mypipe4[0]); } else iF (n4>0){ A+=3;
close mypipe1[0]; write(mypipe4[1];
printF...
close(mypipe4[1]);
else iF (n2>0){ closemypipe[1];
read mypipe[0], &A, sizeof(A));
printF...
close mypipe4[1];
printF("mpampas(%d) received value: %d\n"
getpid(), A);
close mypipe3[0];
```