Λειτουργικά Συστήματα

2η Εργαστηριακή Άσκηση

Μηνάς Δημοσθένης Α.Μ. 1059602 email st1059602@ceid.upatras.gr

Γεωργιάδης Χρήστος ΑΜ 1059579 email st1059579@ceid.upatras.gr

Μακρής Γιώργος ΑΜ 1059679 email st1059679@ceid.upatras.gr

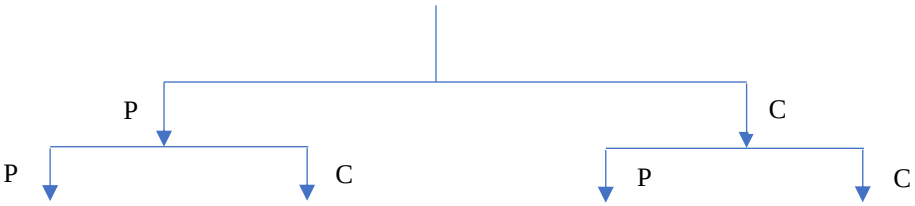
Μέρος 1

Ερώτημα Α

Απάντηση

i.

Μετά την πάροδο 10 sec θα έχουν δημιουργηθεί 4 συνολικά διεργασίες σύμφωνα με το ακόλουθο σχήμα:



- Αρχικά η πρώτη διεργασία που ξεκινά εκτελεί τη fork() και δημιουργεί ένα αντίγραφο του εαυτού της που εκτελείται παράλληλα με αυτή/
- Η καθεμία από τις 2 διεργασίες που έχουν δημιουργηθεί θα εκτελέσει τη συνάρτηση fork() και θα προκύψουν συνολικά 4 διεργασίες 10 sec μετά την έναρξη του προγράμματος

ii.

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
        int pid1;
        int pid2;

        pid1 = fork();

        if (pid1 < 0)
                printf("Could not create any child\n");
        else
                if (pid1>0)
                {
```

```
                printf("1.My PID is %d \n", getpid());
                printf("1. My Parent's PID is %d \n", getppid());
        }
        else
                if (pid1==0)
                {
                        printf("2.My PID is %d \n", getpid());
                        printf("2.My Parent's PID is %d \n", getppid());
                }
    pid2 = fork();
    if (pid2 < 0)
        printf("Could not create any child\n");
    else
        if ((pid1 < 0) && (pid2 < 0))
                kill(pid1,9);
        else
                if (pid2>0)
                {
                        printf("3.My PID is %d \n", getpid());
                        printf("3.My Parent's PID is %d \n", getppid());
                }
                else
                        if (pid2==0)
                        {
                                printf("4.My PID is %d \n", getpid());
                                printf("4.My Parent's PID is %d \n", getppid());
                        }

    sleep(20);
    return (0);
}
```

Αποτελέσματα Εκτέλεσης

1.My PID is 3200

1. My Parent's PID is 2234

3.My PID is 3200

3.My Parent's PID is 2234

2.My PID is 3201

2.My Parent's PID is 3200

3.My PID is 3201

3.My Parent's PID is 3200

4.My PID is 3202

4.My Parent's PID is 3200

4.My PID is 3203

4.My Parent ‘s PID is 3201

Ερώτημα Β

```
sem_t *mutex;

void heapify(int arr[], int n, int i)

{

  int largest = i; // Initialize largest as root

  int l = 2*i + 1; // left = 2*i + 1

  int r = 2*i + 2; // right = 2*i + 2



  if (l < n && arr[l] > arr[largest])

  largest = l;



  if (r < n && arr[r] > arr[largest])

  largest = r;



  if (largest != i)

  {

  int temp=arr[i];

  arr[i]=arr[largest];

  arr[largest]=temp;



  heapify(arr, n, largest);

  }

}

void heapSort(int arr[], int n)

{

  int i;



  for (i = n / 2 - 1; i >= 0; i--) //Κατασκευή Min heap

   heapify(arr, n, i);



  for (i=n-1; i>=0; i--)
```

```
  {
   int temp=arr[0];
   arr[0]=arr[i];
   arr[i]=temp;


   heapify(arr, i, 0); /*μετατρέπουμε το δεντρο πάλι σε σωρό*/
  }
}

int main()
{
   int i, j;
  mutex= sem_open("mut", O_CREAT | O_EXCL, 0644, 0);

   sem_init(&mutex,0,1) ;

  int shm = shmget(42, 100 * sizeof (int), IPC_CREAT | 0666);
//Δημιουργία shared memory key 42 για καταχώριση 100 ακέραιων

  if (shm < 0)
  {
   perror ("Error in creation of shared memory");
   return 1;
  }

  int *sharheap = shmat(shm,NULL,0);/*Βάζω το δείκτη sharheap να
δείχνει στη 1η θέση της κοινής μνήμης που αντιπροωπεύεται από την shm */

  if (sharheap < (int *) NULL)
  {
   perror ("Error in shmat in shared memory");
   return 1;
  }

  for (i = 0; i < 100; ++i) /* Create children and make them work */
   if (fork()==0) /*Αν είμαστε σε κάποιο παιδί*/
```

```
  {
   sem_wait(&mutex); /*down mutex*/

   srand(getpid());/*αρχικοποιούμε τη γεννήτρια παραγωγής
τυχαίων αριθμών με το id της τρέχουσας διεργασίας ώστε να παραχθούν
διαφορετικές random τιμές*/

   sharheap[i]=rand()%getpid();
   usleep(getpid());
   heapSort(sharheap,100);
   sem_post(&mutex) ;/*up mutex*/
   return 0;
  }


 for (i = 0; i< 100; ++i) /*H parent process περιμένει τον τερματισμό των 100 παιδιών της */
  wait (&i);

 puts("\nFinal Sorted Heap");
  for (i = 0; i < 100; ++i)
          printf("%d\t", sharheap[i]);


 shmdt (sharheap); /*διαγραφή της shared memory*/
 shmctl (shm, IPC_RMID, NULL);
  return 0;
}
```

Αποτελέσματα Εκτέλεσης

```
Final Sorted Heap
23 149 316 369 394 401 822 936 943 1076
1231 1284 1487 1819 1893 1976 2057 2084 2139
2241 2298 2305 2353 2401 2430 2513 2554 2605
2643 2695 2795 2887 2975 3066 3110 3252 3572
3617 3670 3721 3725 3766 3818 3894 4026 4138
4277 4329 4394 4613 4628 4735 4765 5160 5188
5192 5299 5416 5440 5524 5585 5634 5709 5790
5945 5959 6241 6334 6351 6516 6601 6721 6779
6837 6837 6920 6979 7008 7305 7314 7419 7602
7846 7918 7968 8031 8169 8348 8471 8518 8562
8580 8964 8983 8986 9148 9151 9208 9439 9495
```

Ερώτημα Γ
Απάντηση

```
sem_t readmut,writemut;
int store = 0,rcount = 0;

FILE *fptr;
void *reader(void *arg)
{
 sleep(3);

 sem_wait(&readmut);
 rcount = rcount + 1;
 if(rcount==1)
  sem_wait(&writemut);

 sem_post(&readmut);

 fptr = fopen("common.txt","r");

 if(fptr == NULL)
 {
    printf("Error in Read!");
    exit(1);
 }
   fscanf(fptr,"%d",&store);

 printf("\nReader read value %d from file. \n",store);

 fclose(fptr);

 sem_wait(&readmut);

 rcount = rcount - 1;

 if(rcount==0)
  sem_post(&writemut);

 sem_post(&readmut);
}


void *writer(void *arg)
{
```

```
  sem_wait(&writemut);

  store++;

  fptr = fopen("common.txt","w");

  if(fptr == NULL)
  {
     printf("Error in Write r!");
     exit(1);
  }

  fprintf(fptr,"%d",store);

    fclose(fptr);

  printf("Data writen by the writer is %d\n",store);

  sleep(1);

  sem_post(&writemut);
}

  pthread_t readthreads[5],writethreads[1];

  sem_init(&readmut,0,1);

  sem_init(&writemut,0,1);


  for(i=0;i<5;i++)
            pthread_create(&readthreads[i],NULL,reader,(void *)i);

  pthread_create(&writethreads[0],NULL,writer,(void *)i);


  for(i=0;i<5;i++)
   pthread_join(readthreads[i],NULL);

  pthread_join(writethreads[0],NULL);

  return 0;

}
```

Αποτελέσματα Εκτέλεσης

Writer stores value 1 in common file.

Writer1 finished writing.
Number of Readers = 1
Number of Readers = 2
Number of Readers = 3
Number of Readers = 4
Number of Readers = 5

Reader read value 1 from file.
Reader read value 1 from file.
Reader read value 1 from file.
Reader read value 1 from file.
Reader read value 1 from file.
Main terminated

Ερώτημα Δ

Απάντηση

(1)

```
sem_t *shmaforos1, *shmaforos2, *shmaforos3, *shmaforos4;


int main()
{

        shmaforos1 = sem_open ("shmafor1", O_CREAT | O_EXCL, 0644,0);


        shmaforos2 = sem_open ("shmafor2", O_CREAT | O_EXCL,
0644,0);


        shmaforos3 = sem_open ("shmafor3", O_CREAT | O_EXCL,
0644,0);


        shmaforos4 = sem_open ("shmafor4", O_CREAT | O_EXCL,
0644,0);


        if (fork() > 0)
        {
                puts("Process P1 starts\n");

                system("pwd");

                sem_post(shmaforos1);

                wait(NULL);

        }


        else
        {

                puts ("Process P2 starts \n");
                system("pwd");

                sem_post(shmaforos2);

                exit(0);

        }


        if (fork() > 0)
        {

                sem_wait (shmaforos1);

                sem_wait (shmaforos2);
```

```
            puts("Process P3 starts \n");
            system("pwd");
            sem_post (shmaforos3);
            sem_post (shmaforos4);
            wait(NULL);
}
else
{
            sem_wait (shmaforos3);
            puts ("Process P4 starts \n");
            system("pwd");
            exit(0);
}

if (fork()>0)
{
            sem_wait (shmaforos4);
            puts ("Process P5 starts \n");

            system("pwd");
            wait(NULL);


            sem_unlink("shmafor1");
            sem_close(shmaforos1);


            sem_unlink("shmafor2");
            sem_close(shmaforos2);


            sem_unlink("shmafor3");
            sem_close(shmaforos3);


            sem_unlink("shmafor4");
            sem_close(shmaforos4);
}
```

```
}
```

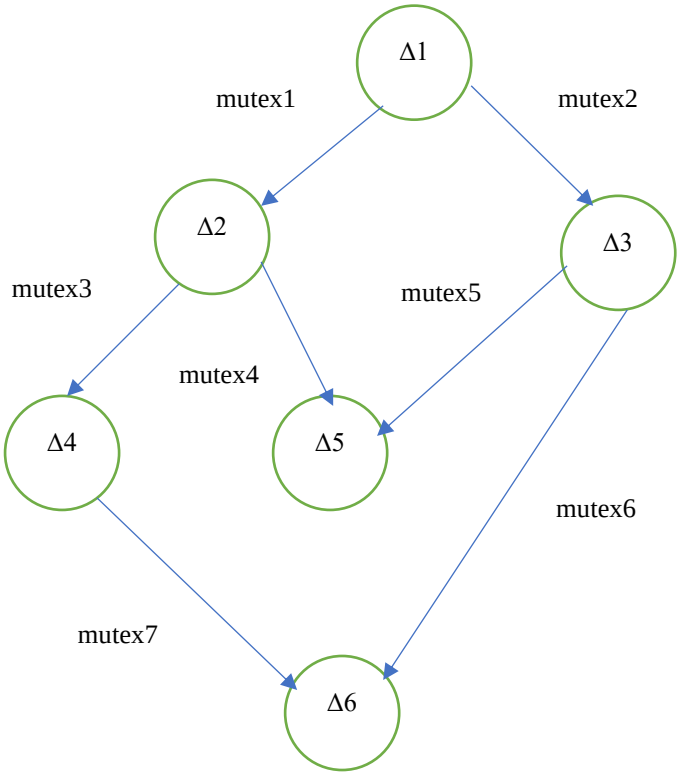
(2)

```
typedef sem_t Semaphore;

Semaphore *shmafor1, *shmafor2, *shmafor3, *shmafor4, *shmafor5,
*shmafor6, *shmafor7;

int main()
{

    shmafor1 = sem_open ("shmafor1", O_CREAT | O_EXCL, 0644, 0);

    shmafor2 = sem_open ("shmafor2", O_CREAT | O_EXCL, 0644, 0);

    shmafor3 = sem_open ("shmafor3", O_CREAT | O_EXCL, 0644, 0);

    shmafor4 = sem_open ("shmafor4", O_CREAT | O_EXCL, 0644, 0);

    shmafor5 = sem_open ("shmafor5", O_CREAT | O_EXCL, 0644, 0);

    shmafor6 = sem_open ("shmafor6", O_CREAT | O_EXCL, 0644, 0);

    shmafor7 = sem_open ("shmafor7", O_CREAT | O_EXCL, 0644, 0);


    if (fork() > 0)
    {

       puts("Process d1 starts\n");

       system("pwd");

       sem_post (shmafor1);

       sem_post (shmafor2);

       wait(NULL);

    }

    else
    {

       sem_wait (shmafor1);

       puts("Process d2 starts \n");

       system("date");

       sem_post (shmafor3);

       sem_post (shmafor4);

       exit(0);

    }



    if (fork() > 0)
    {
       sem_wait (shmafor2);

       puts("Process d3 starts \n");
```

```
    system("ps");
    sem_post (shmafor5);
    sem_post (shmafor6);
    puts ("Process d4 starts \n");
    system("ls");
    sem_post (shmafor7);
    exit(0);
}

if (fork()>0)
{
    sem_wait (shmafor4);
    sem_wait (shmafor5);
    puts ("Process d5 starts \n");
    system("date");
    wait(NULL);

    sem_unlink("shmafor1");
    sem_close(shmafor1);

    sem_unlink("shmafor2");
    sem_close(shmafor2);

    sem_unlink("shmafor3");
    sem_close(shmafor3);

    sem_unlink("shmafor4");
    sem_close(shmafor4);

    sem_unlink("shmafor5");
    sem_close(shmafor5);

    sem_unlink("shmafor6");
```

```
        sem_close(shmafor6);
        sem_unlink("shmafor7");
        sem_close(shmafor7);
    }
    else
    {
        sem_wait (shmafor7);
        sem_wait (shmafor6);
        puts ("Process d6 starts \n");
        system("who");
        exit(0);
    }
}
```

Ο Γράφος διεργασιών είναι ο ακόλουθος:



2ο Μέρος

Ερώτημα Α

Απάντηση

(α) Η μορφή των εικονικών διευθύνσεων είναι

p	d
18	14

Το offset d καθορίζει και το μέγεθος σελίδας που είναι 2^{14}=16.384 bytes. Για να βρούμε πόσες σελίδες καταλαμβάνει η διεργασία διαιρούμε το μέγεθος της διεργασίας με το μέγεθος της σελίδας και προκύπτει: 39.500_{16}=234.752 bytes.

Άρα 234.752:16.384=14,32→15 σελίδες. Συνεπώς η διεργασία καταλαμβάνει συνολικά 15 σελίδες Παρατηρούμε όμως ότι η $15^{η}$ σελίδα είναι γεμάτη μόνο σε ποσοστό 0,32. Άρα το 0,68 της σελίδας παραμένει κενό οπότε η εσωτερική κλασματοποίηση δηλ. ο κενός χώρος εντός της σελίδας είναι 0,68 x 16.384=11.141,12 bytes

(b)

Σύμφωνα με την εκφώνηση οι 5 τελευταίες σελίδες έχουν φορτωθεί στα ακόλουθα φυσικά πλαίσια:

Σελίδα	**Φυσικό Πλαίσιο**
Σελίδα 0	
Σελίδα 1	
…	
Σελίδα 10	16
Σελίδα 11	225
Σελίδα 12	170
Σελίδα 13	35
Σελίδα 14	51

i) 00031958_{16}

0	**0**	**0**	**3**	**1**	**9**	**5**	**8**
0000	0000	0000	0011	00 **01**	**1001**	**0101**	**1000**
00000000000001100					**01100101011000**		
18 bits για τη σελίδα					**14 bits για το offset**		
Σελίδα 12					**Offset 6.488**		

- Τα πρώτα 18 bits παραπέμπουν στη σελίδα 12 που σύμφωνα με την εκφώνηση έχει φορτωθεί στο φυσικό πλαίσιο 170
- Για να βρούμε τη φυσική διεύθυνση στην οποία έχει φορτωθεί η λογική διεύθυνση πολλαπλασιάζουμε τον αριθμό φυσικού πλαισίου (δηλ. του 170) με το μέγεθος του (που είναι επίσης 16.384 bytes)
- Μετά προσθέτουμε το offset που είναι τα 14 τελευταία bits (και είναι ο ακέραιος αριθμός 6488) και προκύπτει η τελική φυσική διεύθυνση που είναι:

$170 \bullet 16.384+6.488=2.791.768$ και σε δεκαεξαδική μορφή $2A9958_{16}$

ii) $0001E800_{16}$

0	**0**	**0**	**1**	**E**	**8**	**0**	**0**
0000	**0000**	**0000**	**0001**	**11 10**	**1000**	**0000**	**0000**
00000000000000111					**10100000000000**		
18 bits για τη σελίδα					**14 bits για το offset**		
Σελίδα 7					**Offset**		

- Τα πρώτα 18 bits παραπέμπουν στη σελίδα 7 που σύμφωνα με την εκφώνηση ΔΕΝ έχει φορτωθεί σε φυσικό πλαίσιο μνήμης (μόνο οι 5 τελευταίες σελίδες έχουν φορτωθεί σε φυσικά πλαίσια μνήμης) άρα προκαλείται page fault

Ερώτημα Β

Απάντηση

(α)

- Το μέγεθος του εικονικού χώρου διευθύνσεων είναι 2^{32} bytes
- Το μέγεθος του κάθε τμήματος σε bytes είναι: $16 \bullet 1.024 \bullet 1.024 = 2^{24}$ bytes
- Ο αριθμός τμημάτων για μια διεργασία είναι: 2^{32} bytes:2^{24} bytes = 2^8 =256 τμήματα

(β)

(i) $0B00042A_{16}$

Η μορφή των λογικών διευθύνσεων στην τμηματοποίηση είναι:

s	**d**

Στη συγκεκριμένη άσκηση η μορφή λογικών διευθύνσεων είναι:

s	**d**
8	24

όπου s το τμήμα και d η μετατόπιση από την αρχική του διεύθυνση. Μετατρέπουμε τη λογική διεύθυνση που δίνεται σε αυτή τη μορφή:

s		**d**					
0	**B**	**0**	**0**	**0**	**4**	**2**	**A**
0000	1011	0000	0000	0000	0100	0010	1010
11		**1066**					

Τα πρώτα 8 bits από αριστερά παραπέμπουν στο τμήμα μα αριθμό 11, ενώ τα τελευταία 24 bit παραπέμπουν στο offset 1066. Άρα η λογική διεύθυνση που δίνεται σε μορφή (s,d) είναι [11, 1066] δηλαδή τμήμα 11 με offset 1.066. Για να μετατρέψουμε την εικονική διεύθυνση σε φυσική εξετάζουμε πρώτα αν το d<limit για το συγκεκριμένο τμήμα που βρισκόμαστε. Επειδή d=1.066<limit=1.230 η λογική διεύθυνση είναι έγκυρη δηλαδή έχει φορτωθεί σε κάποιο τμήμα (segment). Η φυσική διεύθυνση είναι base address +offset=9.050+1.066=10.116. Τη μετατρέπουμε σε δεκαεξαδική και προκύπτει η **φυσική διεύθυνση 2.784_{16}**

(ii) $02000B6D_{16}$

s		**d**					
0	**2**	**0**	**0**	**0**	**B**	**6**	**D**
0000	0010	**0000**	**0000**	**0000**	**1011**	**0110**	**1101**
2		**2915**					

Άρα η λογική διεύθυνση που δίνεται σε μορφή (s,d) είναι [2, 2925]. Για να μετατρέψουμε την εικονική διεύθυνση σε φυσική εξετάζουμε πρώτα αν το d<limit για το συγκεκριμένο πλαίσιο. Επειδή d=2.915>limit=1.290 άρα η λογική διεύθυνση ΔΕΝ είναι έγκυρη, δηλαδή ΔΕΝ έχει φορτωθεί σε κάποιο τμήμα (segment). άρα έχουμε **ΣΦΑΛΜΑ ΤΜΗΜΑΤΟΣ (Segment fault)**

Ερώτημα Γ

Απάντηση

(α)

Οι λογικές διευθύνσεις στη σελιδοποιημένη τμηματοποίηση (paged segmentation) έχουν την ακόλουθη μορφή

s	p	d

όπου το s είναι το τμήμα (segment), το p είναι η σελίδα (page) και το d είναι το offset

- Έχουμε από πριν 256 τμήματα άρα τα bits για το segment είναι τα πρώτα 8 bits
- Το μέγεθος σελίδας είναι 512 bytes άρα το offset είναι τα τελευταία 9 bits
- Άρα τα bits που απομένουν για το πλήθος σελίδων είναι 15 δηλαδή υπάρχουν 2^{15} σελίδες σε κάθε πίνακα σελίδων
- Η μορφή των λογικών διευθύνσεων είναι συνεπώς:

s	**p**	**d**
8	**15**	**9**

(β)

Μια διεργασία αποτελείται από 2^8 τμήματα καθένα από τα οποία περιλαμβάνει 2^{15} σελίδες. Άρα ο μέγιστος αριθμός σελίδων μιας διεργασίας είναι 2^8 τμήματα $\bullet 2^{15}$ σελίδες/τμήμα $=2^{23}$ σελίδες

(γ)

(i) $010004CF_{16}$

s		**p**				**d**		
8		**15**				**9**		
0000	0001	0000	0000	0000	010	**0**	**1100**	**1111**
Segment=1		**Page=2**				**Offset=207**		

Πηγαίνουμε στο τμήμα (segment) με αριθμό 1 και επιλέγουμε τη σελίδα με αριθμό 2 που έχει φορτωθεί στο φυσικό πλαίσιο $0B0B_{16}$ =2.827. Αυτό το φυσικό πλαίσιο αρχίζει στη διεύθυνση $2.827 \bullet 512=1.447.424$. Σε αυτή την αρχική διεύθυνση προσθέτουμε το offset που είναι 207 και προκύπτει η τελική φυσική διεύθυνση που είναι:

$2.827 \bullet 512+207=1.447.631$ και σε δεκαεξαδική μορφή είναι $1616CF_{16}$

Αν σε αυτό το φυσικό πλαίσιο προσθέσουμε το offset που είναι το CF προκύπτει η φυσική διεύθυνση 0B0BCF

(ii)

-010009FF$_{16}$

s		**p**				**d**		
8		**15**				**9**		
0000	0001	0000	0000	0000	100	**1**	**1111**	**1111**
Segment=1		**Page=4**				**Offset=511**		

Η λογική διεύθυνση 010009FF$_{16}$ έχει φορτωθεί στο segment (τμήμα 1) και στη σελίδα 4. Η σελίδα 4 δεν έχει φορτωθεί σε φυσικό πλαίσιο μνήμης. **Άρα βάζουμε – στο αντίστοιχο ?**

-000003F0$_{16}$

s		**p**				**d**		
8		**15**				**9**		
0000	0000	0000	0000	0000	001	**1**	**1111**	**0000**
Segment=0		**Page=1**				**Offset=240**		

Η φυσική διεύθυνση E0E1F0$_{16}$ είναι της μορφής

1110	0000	1110	000	**1**	**1111**	**0000**
Frame=7070				**Offset=240**		

Τα πρώτα 15 bits προσδιορίζουν το φυσικό πλαίσιο μνήμης και τα τελευταία 9 bits προσδιορίζουν το offset. Συνεπώς προκύπτει ότι ο αριθμός φυσικού πλαισίου είναι:

111000001110000=**7070$_{16}$**

Άρα θέτουμε 7070 στο αντίστοιχο ? για τη σελίδα 1 του Πίνακα Σελίδων Τμήματος 0

Ερώτημα Δ

Απάντηση

	3	5	8	1	8	7	5	1	8	2	4	2	7	3	6	4	7	5	3	7
0	**3**	**3**	**3**	**3**	**3**	**7**	**7**	**7**	**7**	**2**	**2**	**2**	**2**	**2**	**2**	**4**	**4**	**4**	**4**	**4**
1		**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**4**	**4**	**4**	**4**	**6**	**6**	**6**	**6**	**3**	**3**
2			**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**3**	**3**	**3**	**3**	**5**	**5**	**5**
3				**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**
	F	**F**	**F**	**F**	**H**	**F**	**H**	**H**	**H**	**F**	**F**	**H**	**F**	**F**	**F**	**F**	**H**	**F**	**F**	**H**