

Процеси

проф. д-р инж. Христо Вълчанов

<http://cs.tu-varna.bg>

Основни концепции

- **Процес** – програма по време на нейното изпълнение.
- Включва множество компонента:
 - Програмен код (text section).
 - Текущата му активност, включваща регистрите на процесора.
 - Стек.
 - Даннова част с глобални променливи.
 - Динамично заемана памет (Heap).

Програма и процес

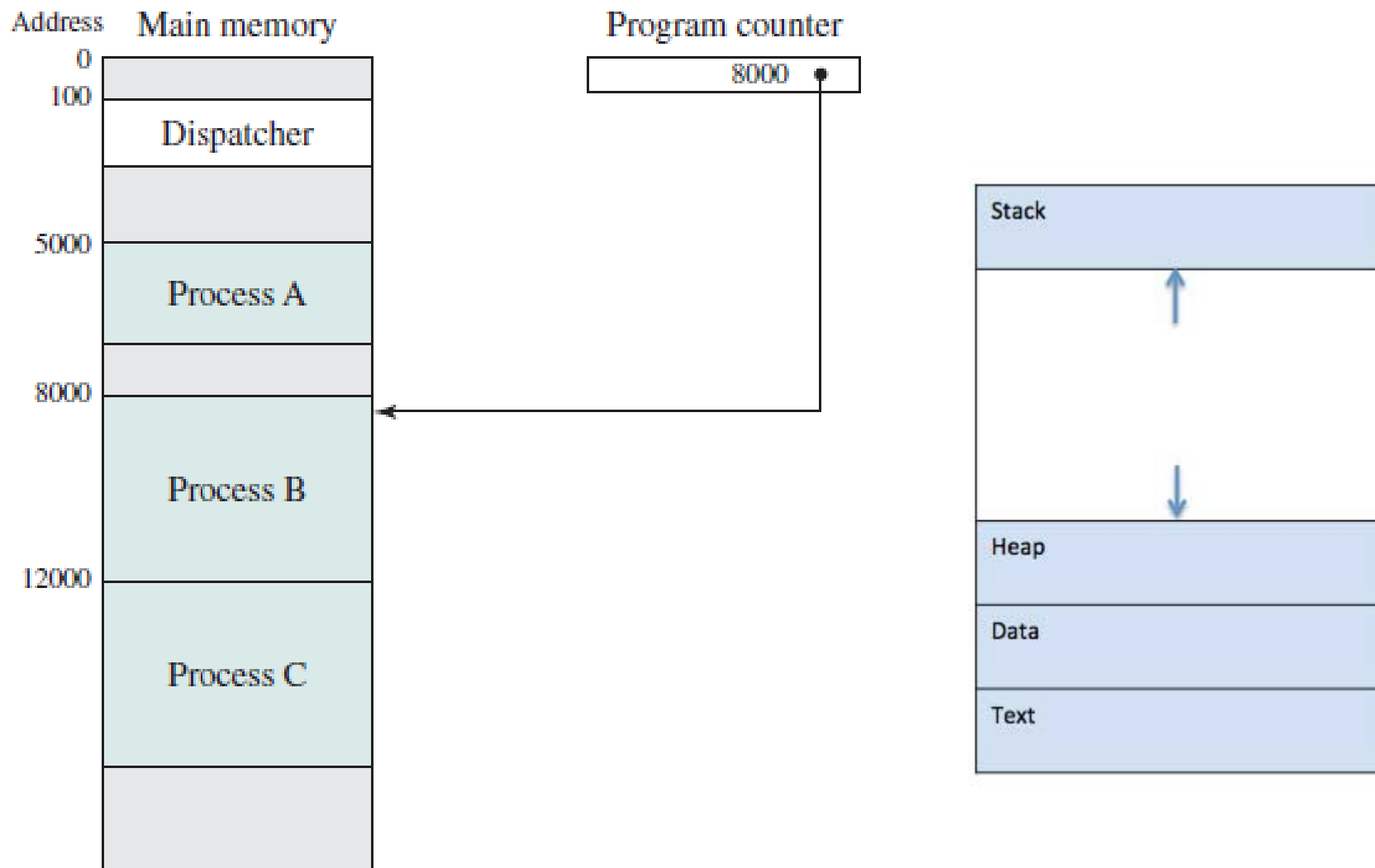
- **Програма** – пасивен компонент, съхранен на диска (изпълним файл).
- **Процес** – активен компонент.
- Програмата става процес, когато изпълнимият файл се зареди в паметта.
- Една програма може да се изпълнява като множество процеси.

Блок за управление на процеса

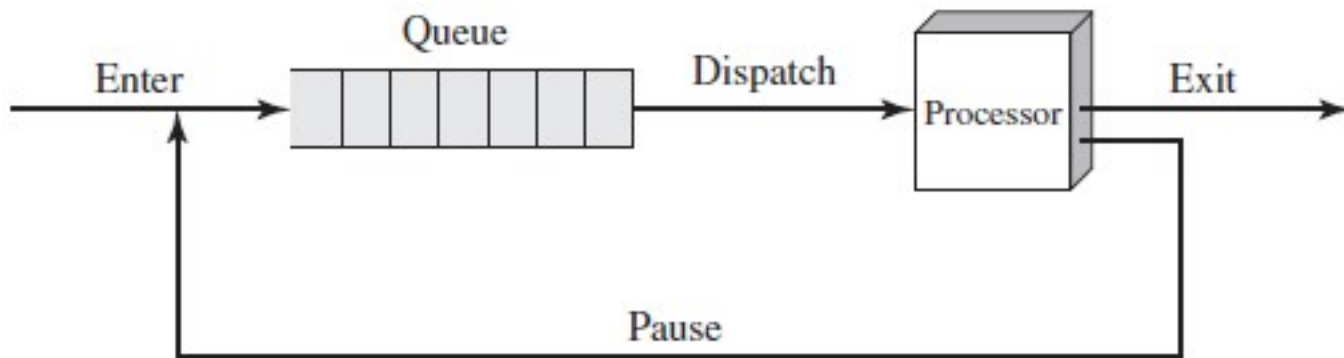
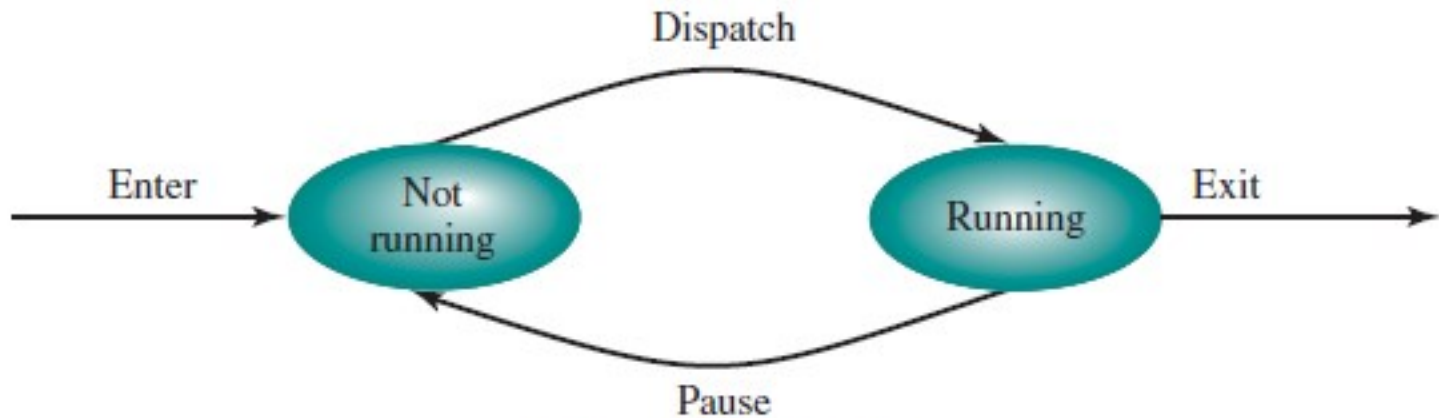
Информация, асоциирана с всеки процес (Process Control Block – PCB).

Process ID
State
Pointer
Priority
Program counter
CPU registers
I/O information
Accounting information
etc....

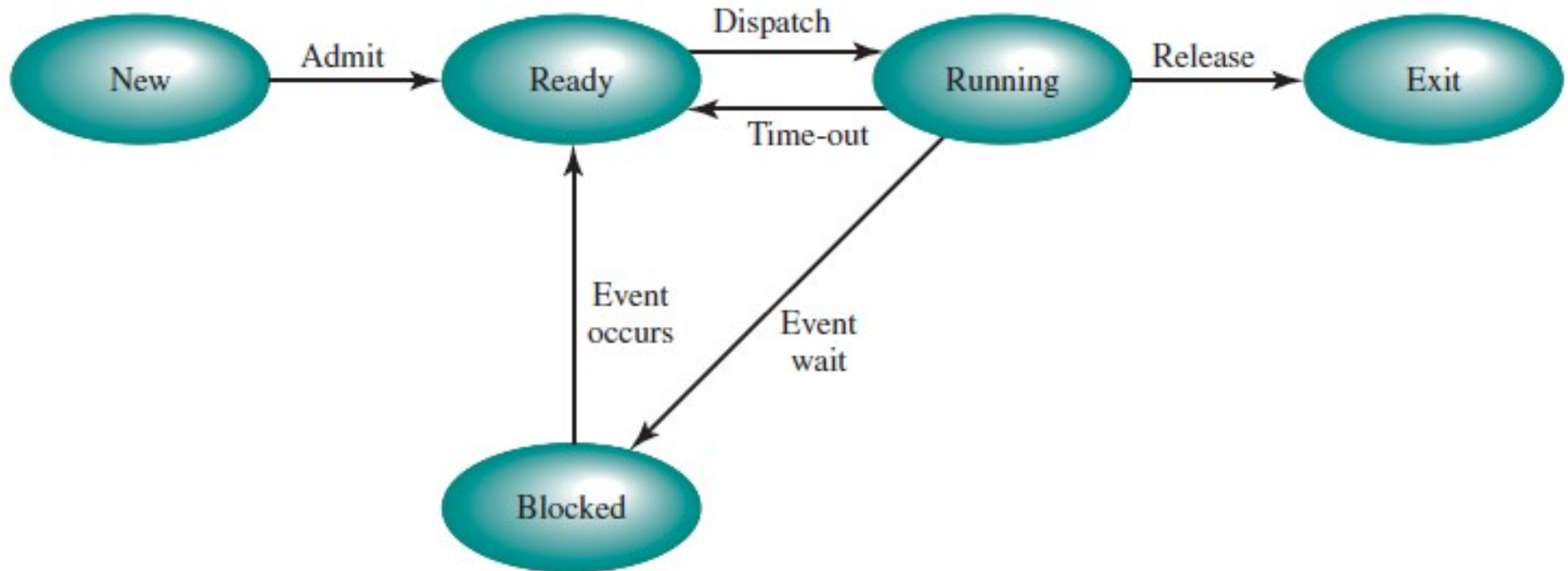
Процеси в паметта



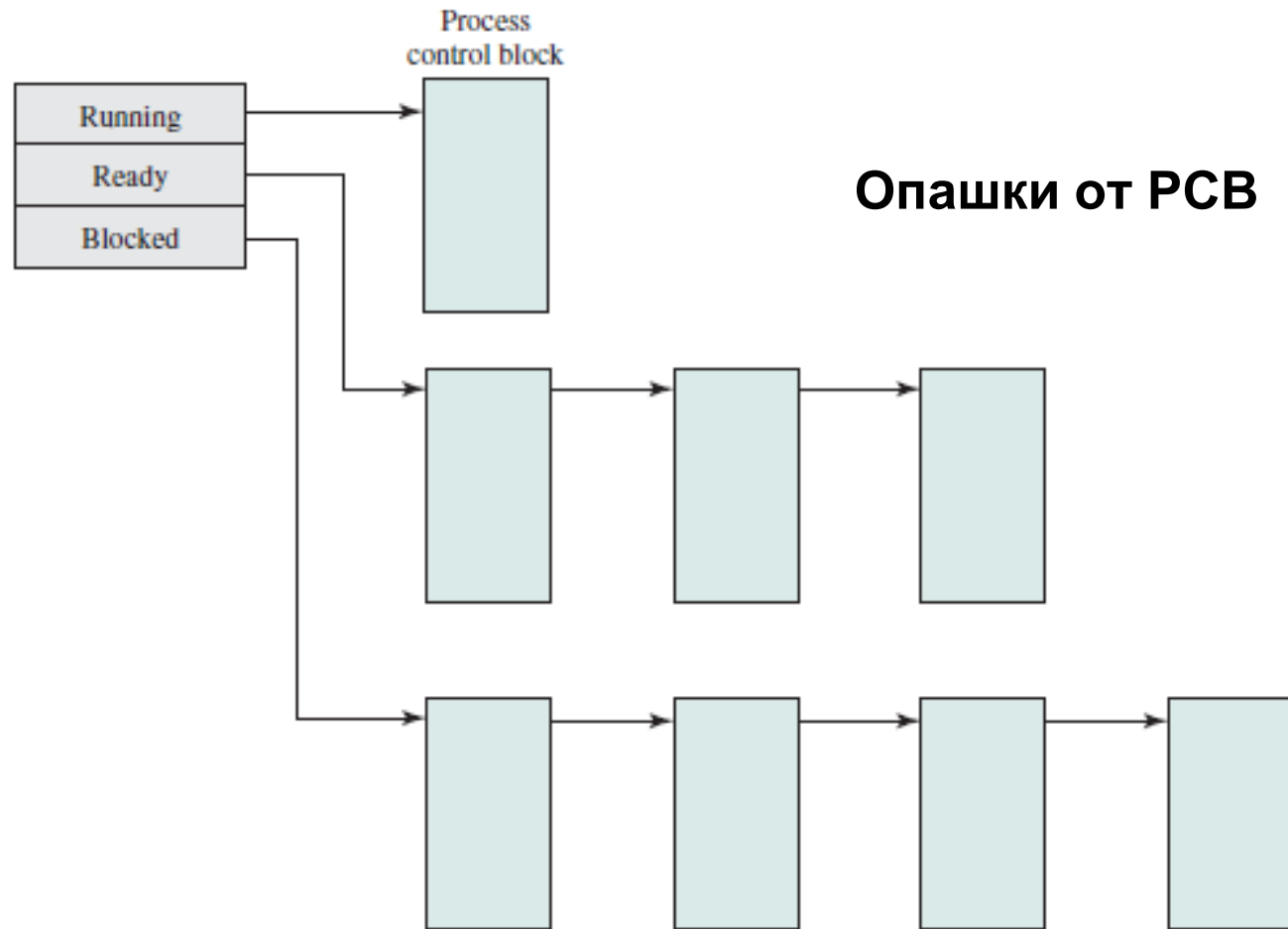
Модел с две състояния на процесите



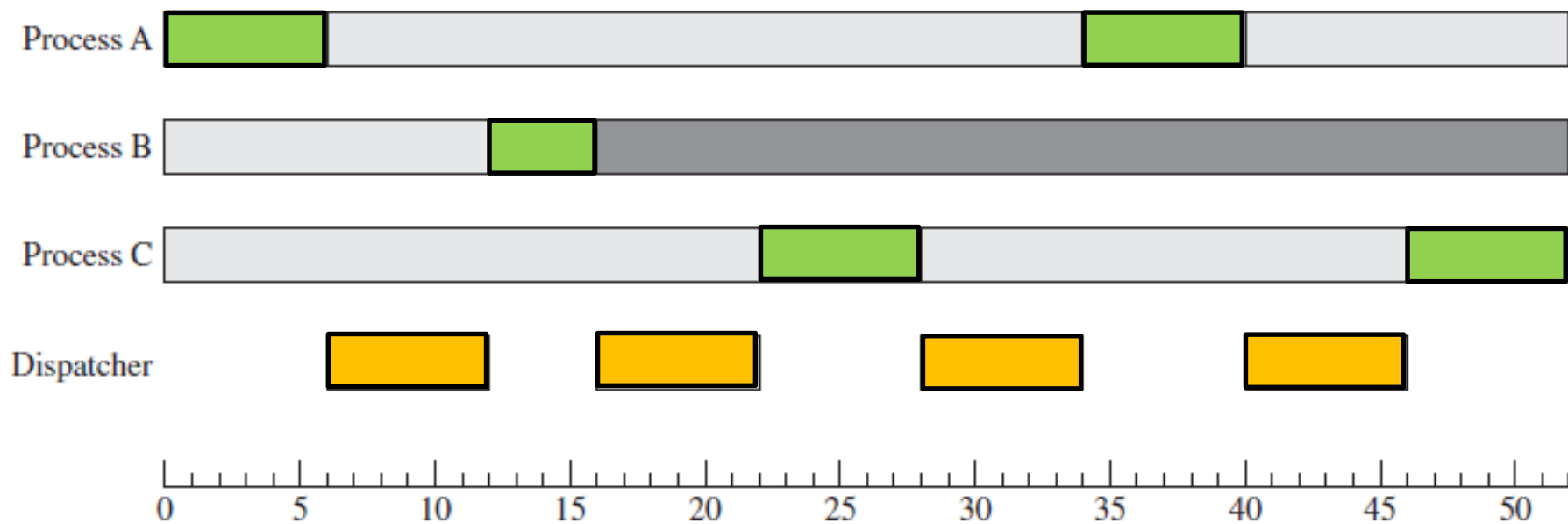
Модел с пет състояния на процесите



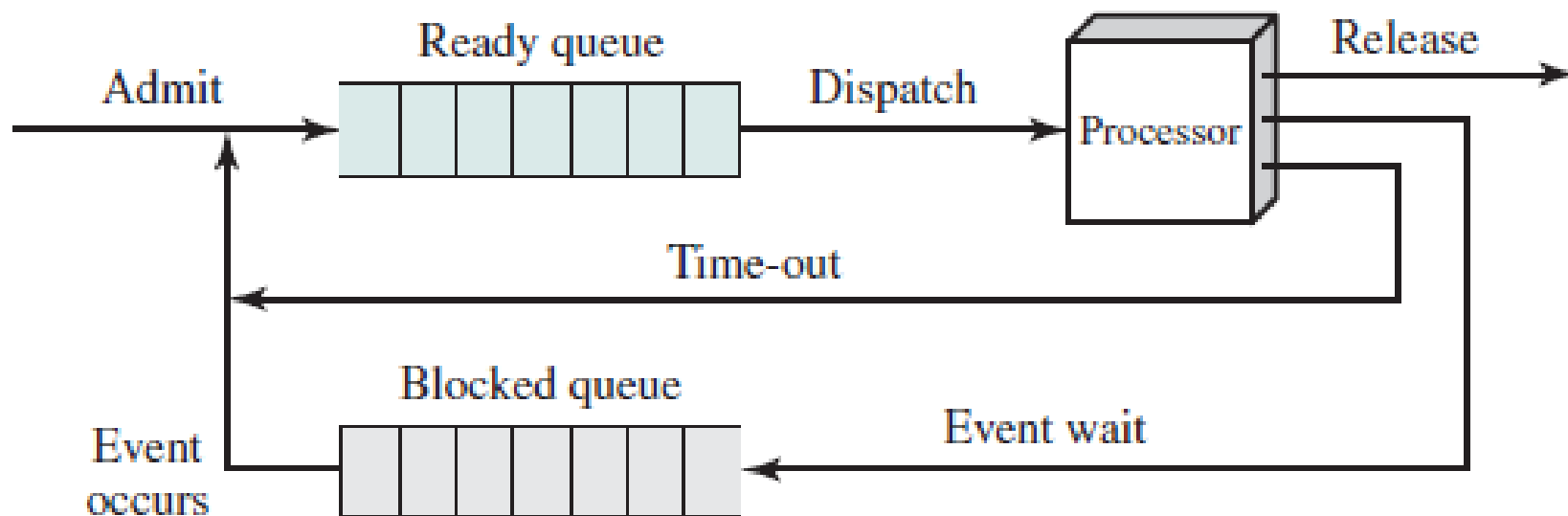
Организация на състоянията на процесите



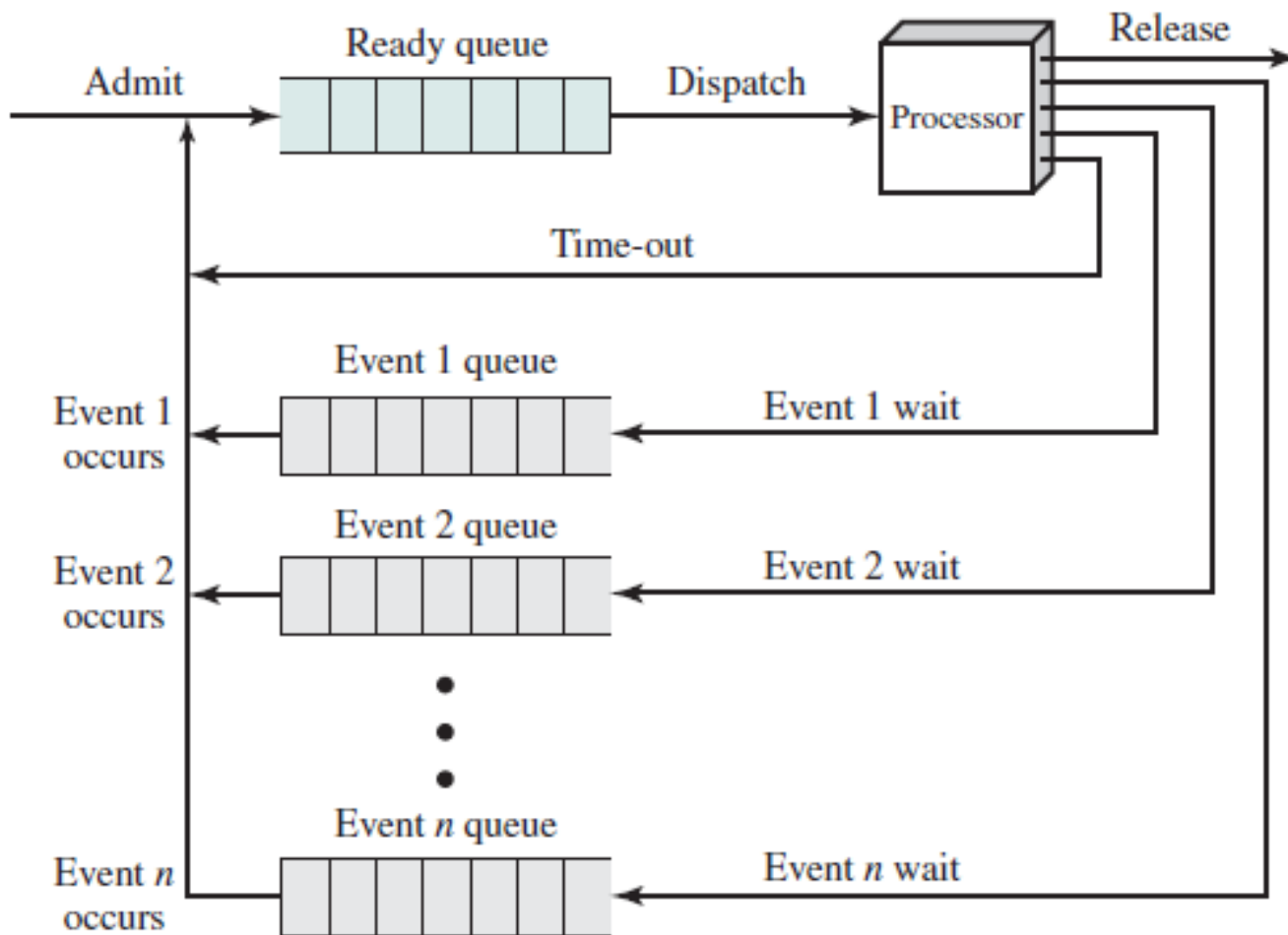
Изпълнение на процесите



Единична опашка на блокирани процеси



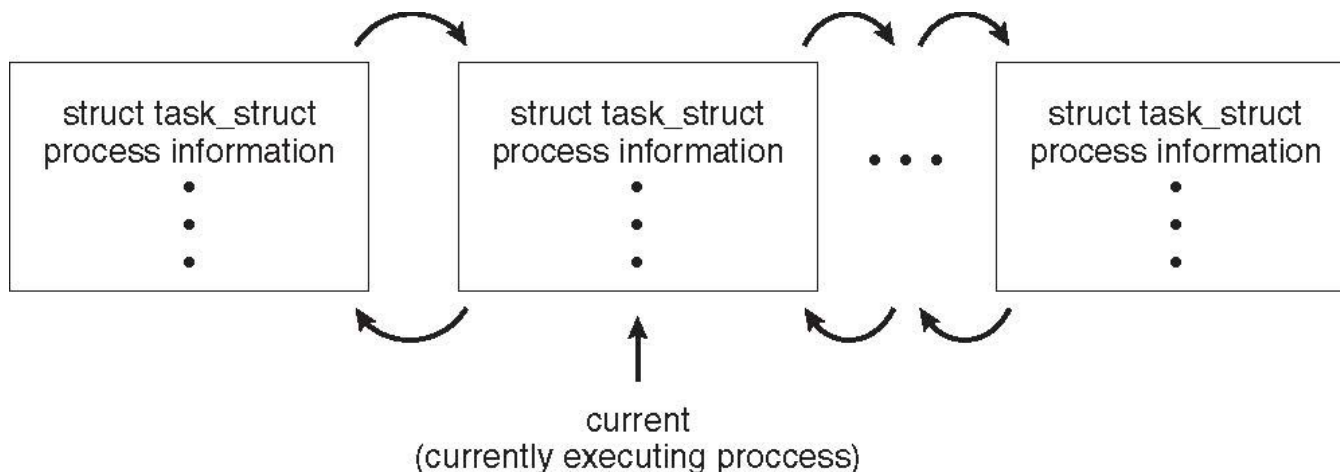
Множество опашки на блокирани процеси



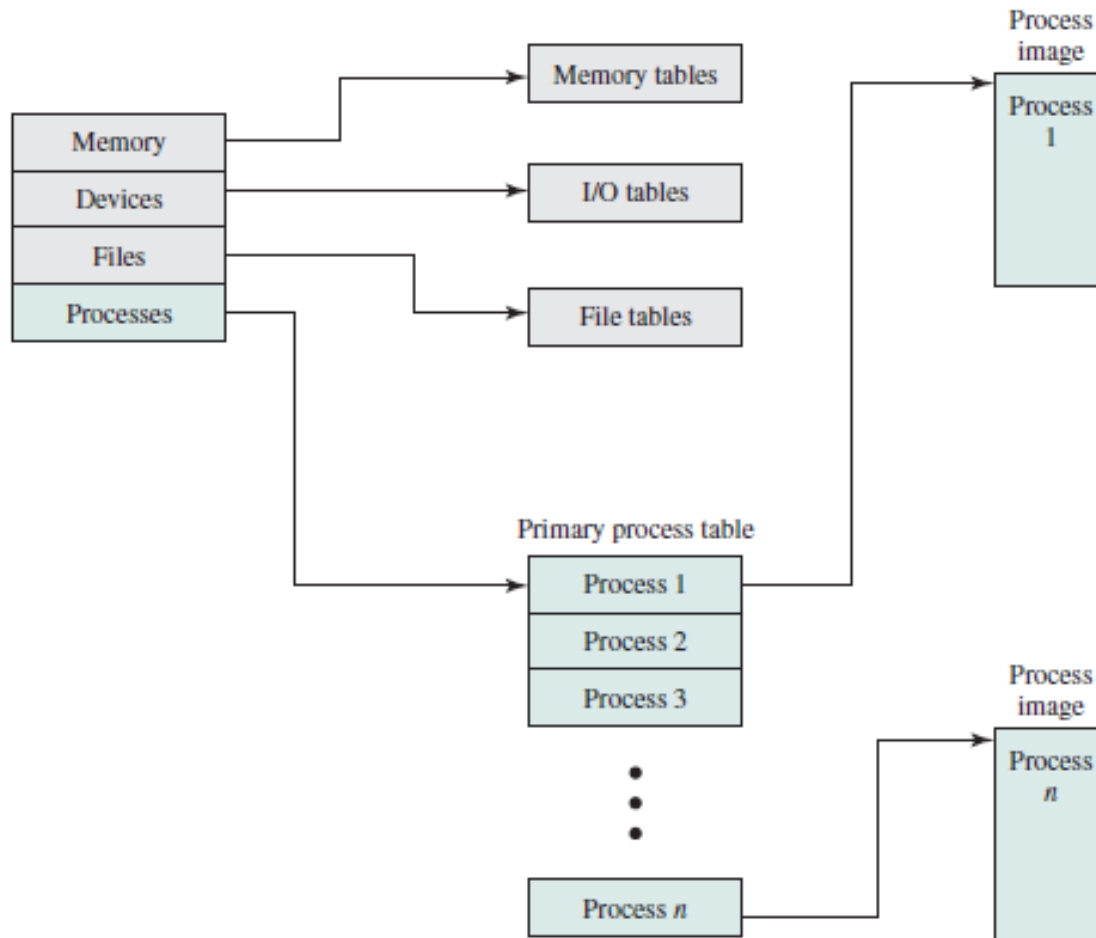
Реализация на процес в Linux

task_struct

```
pid_t pid; /* process identifier */
long state; /* state of the process */
unsigned int time_slice; /* scheduling information */
struct task_struct *parent; /* this process's parent */
struct list_head children; /* this process's children */
struct files_struct *files; /* list of open files */
struct mm_struct *mm; /* address space of this process */
```



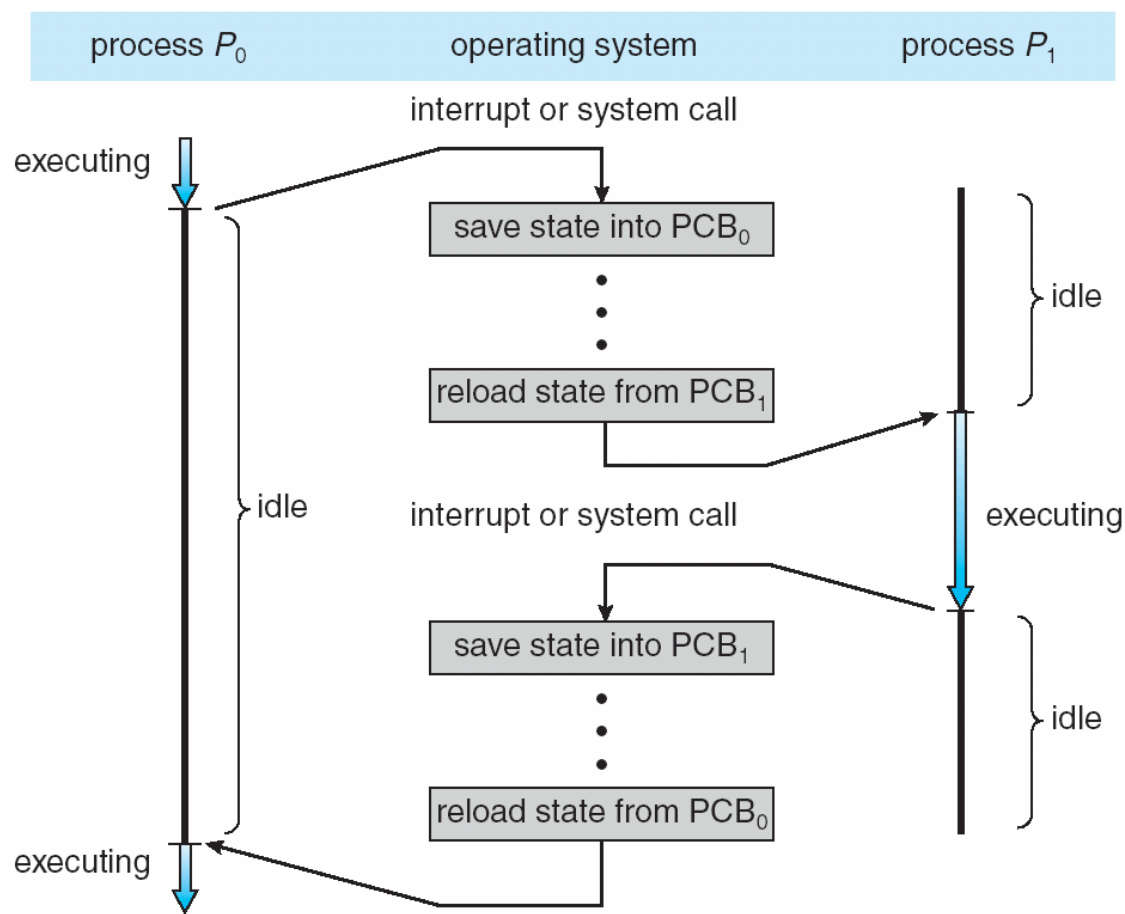
Структури данни на ОС за управление



Превключване контекста на процесите

- При превключване към друг процес системата трябва да съхрани **състоянието** на стария процес и да зареди състоянието на новия чрез превключване на контекста.
- Контекстът на процес се съхранява в PCB.
- Превключването на контекста води до системни разходи (**overhead**).

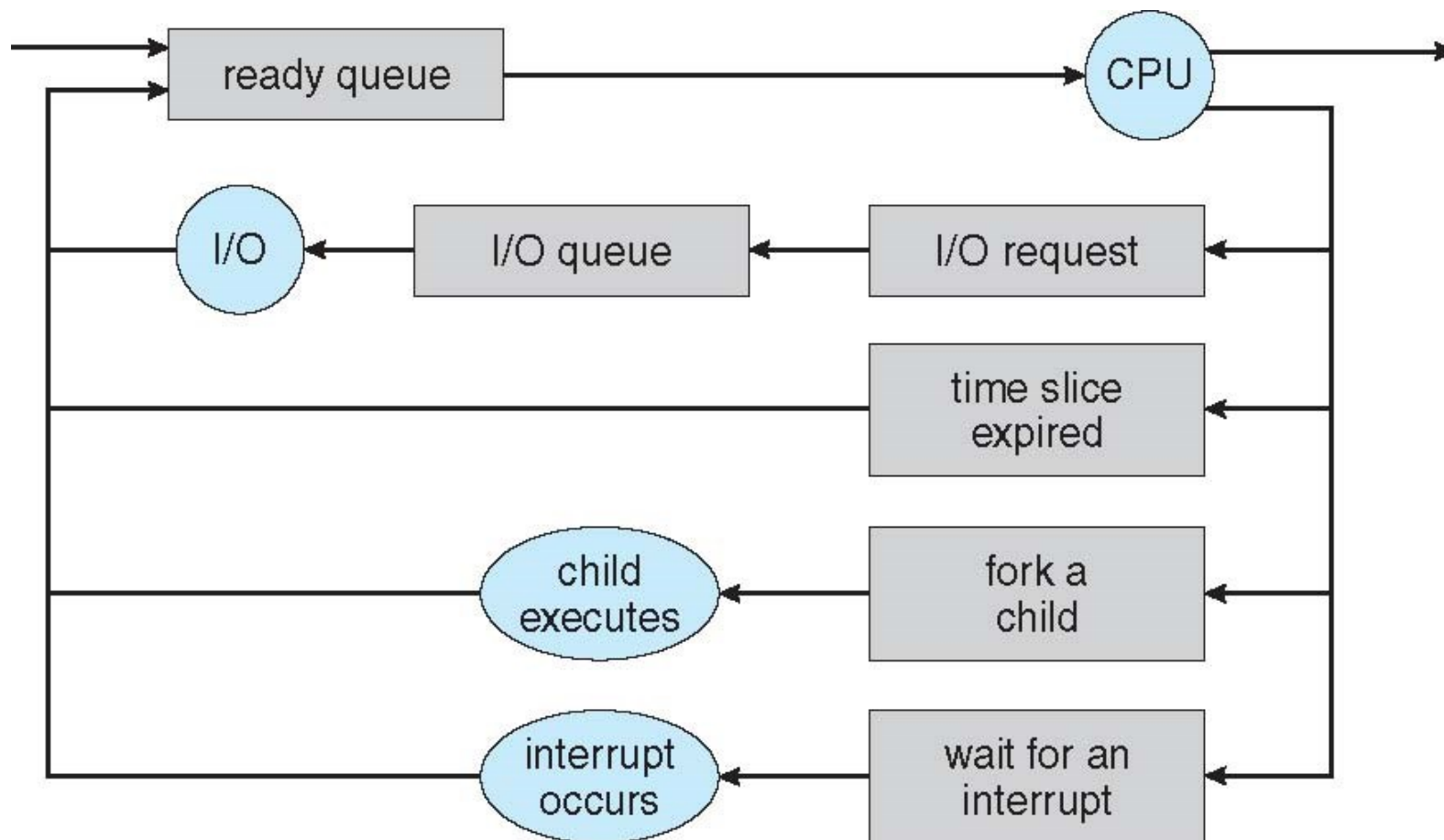
Превключване на процесора между процесите



Планиране на процесите

- **Планировчик (scheduler)** – избира за изпълнение върху процесора един измежду няколко готови процеса.
- Поддържа опашки от процеси.

Диаграма на превключванията на процесите



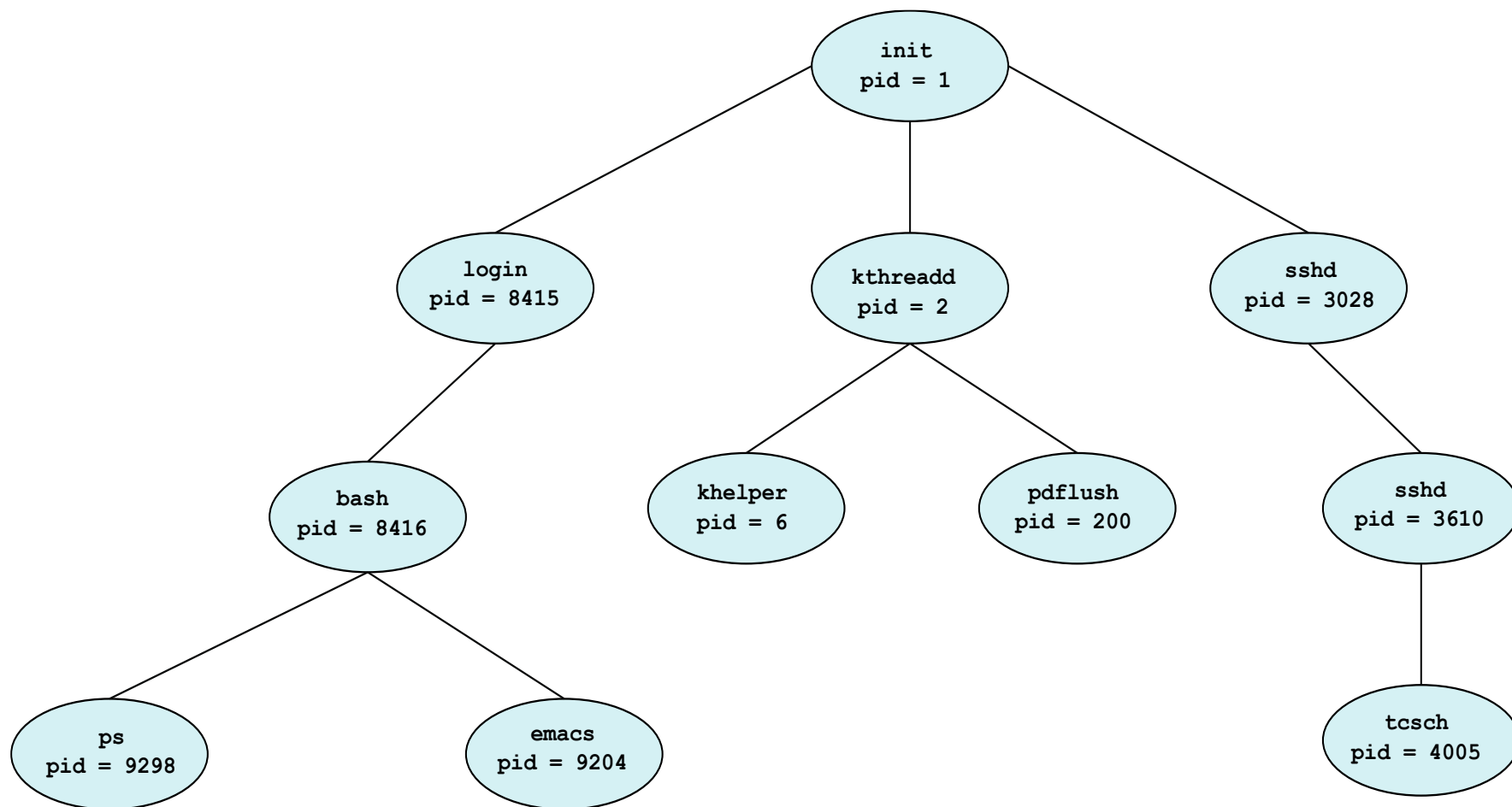
Основни операции върху процесите

- Създаване;
- Завършване.

Създаване на процес

- Създаващият процес – **родител (parent)**.
- Създаденият процес – **наследник (child)**.
- Всеки наследник може да създава процеси – формира се дърво на процесите (**Process tree**).
- Споделяне на ресурси.
- Изпълнение.

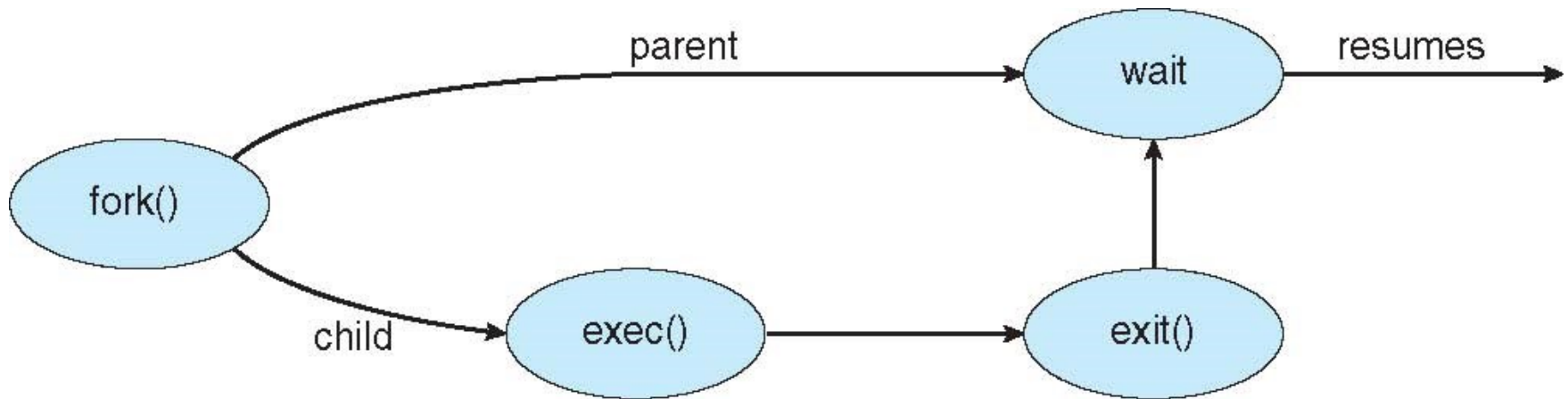
Дърво на процесите в Linux



Създаване на процес в Linux

- Адресно пространство:
 - Наследникът се явява копие на родителя;
 - Наследникът зарежда програма за изпълнение.
- Unix:
 - Създаване на процес с **fork()**;
 - Системното извикване **exec()** се използва след *fork()* за замяна на паметта на родителя с нова програма.
- Родителят може да чака завършването на наследника - **wait()**.

Създаване на процес



Създаване на процес - Linux

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

int main()
{
    pid_t pid;

    /* fork a child process */
    pid = fork();

    if (pid < 0) { /* error occurred */
        fprintf(stderr, "Fork Failed");
        return 1;
    }
    else if (pid == 0) { /* child process */
        execlp("/bin/ls", "ls", NULL);
    }
    else { /* parent process */
        /* parent will wait for the child to complete */
        wait(NULL);
        printf("Child Complete");
    }

    return 0;
}
```

Създаване на процес - Linux

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

parent

```
int main()
{
    pid_t pid;

    /* fork a child process */
    pid = fork();

    if (pid < 0) { /* error occurred */
        fprintf(stderr, "Fork Failed");
        return 1;
    }
    else if (pid == 0) { /* child process */
        execlp("/bin/ls", "ls", NULL);
    }
    else { /* parent process */
        /* parent will wait for the child to
        wait(NULL);
        printf("Child Complete");
        }

    return 0;
}
```

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

child

```
int main()
{
    pid_t pid;

    /* fork a child process */
    pid = fork();

    if (pid < 0) { /* error occurred */
        fprintf(stderr, "Fork Failed");
        return 1;
    }
    else if (pid == 0) { /* child process */
        execlp("/bin/ls", "ls", NULL);
    }
    else { /* parent process */
        /* parent will wait for the child to
        wait(NULL);
        printf("Child Complete");
        }

    return 0;
}
```


Създаване на процес - Linux

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

parent

```
int main()
{
    pid_t pid;

    /* fork a child process */
    pid = fork();

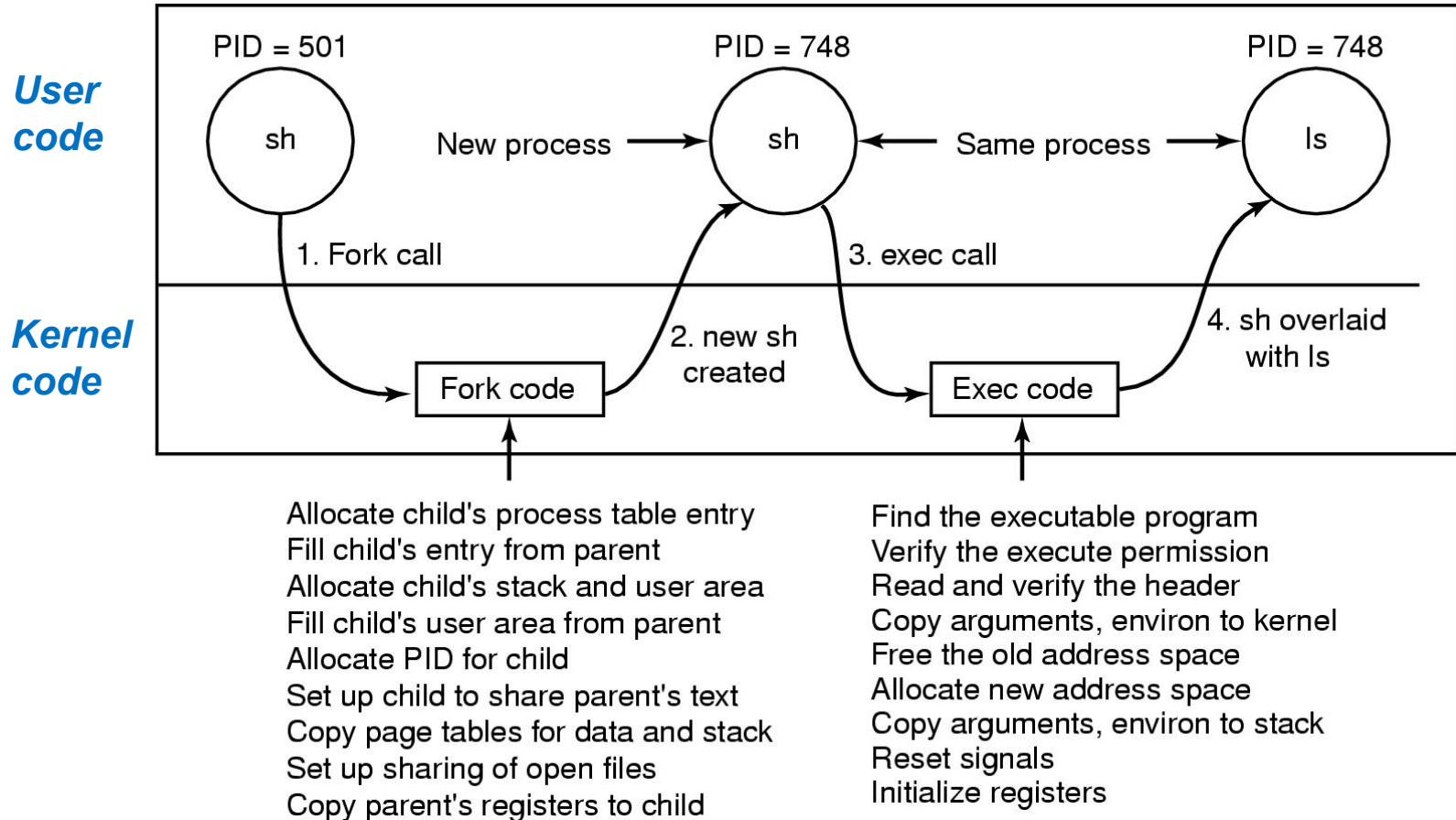
    if (pid < 0) { /* error occurred */
        fprintf(stderr, "Fork Failed");
        return 1;
    }
    else if (pid == 0) { /* child process */
        execlp("/bin/ls", "ls", NULL);
    }
    else { /* parent process */
        /* parent will wait for the child t
        wait(NULL);
        printf("Child Complete");
    }

    return 0;
}
```

child

КОД НА /bin/ls

Изпълнение на команда /s



Operating Systems, 2013, Meni Adler, Michael Elhadad & Amnon Meisels

Създаване на процес - Windows

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

int main(VOID)
{
    STARTUPINFO si;
    PROCESS_INFORMATION pi;

    /* allocate memory */
    ZeroMemory(&si, sizeof(si));
    si.cb = sizeof(si);
    ZeroMemory(&pi, sizeof(pi));

    /* create child process */
    if (!CreateProcess(NULL, /* use command line */
        "C:\\\\WINDOWS\\system32\\mspaint.exe", /* command */
        NULL, /* don't inherit process handle */
        NULL, /* don't inherit thread handle */
        FALSE, /* disable handle inheritance */
        0, /* no creation flags */
        NULL, /* use parent's environment block */
        NULL, /* use parent's existing directory */
        &si,
        &pi))
    {
        fprintf(stderr, "Create Process Failed");
        return -1;
    }
    /* parent will wait for the child to complete */
    WaitForSingleObject(pi.hProcess, INFINITE);
    printf("Child Complete");

    /* close handles */
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
}
```

Завършване на процес

- Процес завършва нормално след изпълнение на последния оператор от програмата чрез извикване на **exit()**.
 - В тази точка процесът връща код на резултат от своето завършване на родителя си.
 - Всички заети ресурси от процеса се освобождават от ОС.

Извеждане на процесите под Linux

```
194.141.30.86 - PuTTY
root@LinuxOS:~# ps ax
  PID TTY          STAT       TIME COMMAND
    1 ?           Ss        2:24   init [3]
    2 ?           S         0:03   [kthreadd]
    3 ?           S         5:49   [ksoftirqd/0]
    5 ?           S<        0:00   [kworker/0:0H]
    7 ?           S      357:28   [rcu_sched]
    8 ?           S         0:00   [rcu_bh]
    9 ?           S         0:00   [migration/0]
   10 ?           S         0:14   [migration/1]
   11 ?           S         0:00   [ksoftirqd/1]
   13 ?           S<        0:00   [kworker/1:0H]
   14 ?           S         0:02   [migration/2]
   15 ?           S         0:00   [ksoftirqd/2]
   17 ?           S<        0:00   [kworker/2:0H]
   18 ?           S         0:00   [migration/3]
   19 ?           S         0:00   [ksoftirqd/3]
   21 ?           S<        0:00   [kworker/3:0H]
   22 ?           S         0:00   [migration/4]
   23 ?           S         0:00   [ksoftirqd/4]
   25 ?           S<        0:00   [kworker/4:0H]
   26 ?           S         0:00   [migration/5]
   27 ?           S         0:00   [ksoftirqd/5]
   29 ?           S<        0:00   [kworker/5:0H]
   30 ?           S         0:00   [migration/6]
   31 ?           S         0:00   [ksoftirqd/6]
   33 ?           S<        0:00   [kworker/6:0H]
   34 ?           S         0:00   [migration/7]
   35 ?           S         0:00   [ksoftirqd/7]
   37 ?           S<        0:00   [kworker/7:0H]
   38 ?           S         0:00   [kdevtmpfs]
   39 ?           S<        0:00   [netns]
   40 ?           S<        0:00   [perf]
   41 ?           S<        0:00   [writeback]
   42 ?           SN        0:40   [khugepaged]
   43 ?           S<        0:00   [crypto]
```

ps ax

Извеждане на процесите под Linux

```
194.141.30.86 - PuTTY
top - 18:48:17 up 66 days, 2:38, 2 users, load average: 0.00, 0.00, 0.00
Tasks: 219 total, 1 running, 218 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.2 us, 0.1 sy, 0.0 ni, 99.7 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 16402796 total, 14556984 free, 256608 used, 1589204 buff/cache
KiB Swap: 0 total, 0 free, 0 used. 15607500 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR  S  %CPU  %MEM    TIME+  COMMAND
    7 root        20   0       0       0       0  S   0.7   0.0 357:29.17 rcu_sched
24331 root        20   0    20540    2944   2448  R   0.7   0.0  0:00.09 top
24335 sshd        20   0    30020     436       0  S   0.3   0.0  0:00.01 sshd
    1 root        20   0     4372    1532   1428  S   0.0   0.0  2:24.87 init
    2 root        20   0       0       0       0  S   0.0   0.0  0:03.29 kthreadd
    3 root        20   0       0       0       0  S   0.0   0.0  5:49.99 ksoftirqd/0
    5 root         0 -20       0       0       0  S   0.0   0.0  0:00.00 kworker/0:0H
    8 root        20   0       0       0       0  S   0.0   0.0  0:00.00 rcu_bh
    9 root        rt    0       0       0       0  S   0.0   0.0  0:00.19 migration/0
   10 root        rt    0       0       0       0  S   0.0   0.0  0:14.59 migration/1
   11 root        20   0       0       0       0  S   0.0   0.0  0:00.06 ksoftirqd/1
   13 root         0 -20       0       0       0  S   0.0   0.0  0:00.00 kworker/1:0H
   14 root        rt    0       0       0       0  S   0.0   0.0  0:02.53 migration/2
   15 root        20   0       0       0       0  S   0.0   0.0  0:00.03 ksoftirqd/2
   17 root         0 -20       0       0       0  S   0.0   0.0  0:00.00 kworker/2:0H
   18 root        rt    0       0       0       0  S   0.0   0.0  0:00.41 migration/3
   19 root        20   0       0       0       0  S   0.0   0.0  0:00.06 ksoftirqd/3
   21 root         0 -20       0       0       0  S   0.0   0.0  0:00.00 kworker/3:0H
   22 root        rt    0       0       0       0  S   0.0   0.0  0:00.26 migration/4
   23 root        20   0       0       0       0  S   0.0   0.0  0:00.04 ksoftirqd/4
   25 root         0 -20       0       0       0  S   0.0   0.0  0:00.00 kworker/4:0H
   26 root        rt    0       0       0       0  S   0.0   0.0  0:00.22 migration/5
   27 root        20   0       0       0       0  S   0.0   0.0  0:00.04 ksoftirqd/5
   29 root         0 -20       0       0       0  S   0.0   0.0  0:00.00 kworker/5:0H
   30 root        rt    0       0       0       0  S   0.0   0.0  0:00.21 migration/6
   31 root        20   0       0       0       0  S   0.0   0.0  0:00.04 ksoftirqd/6
   33 root         0 -20       0       0       0  S   0.0   0.0  0:00.00 kworker/6:0H
   34 root        rt    0       0       0       0  S   0.0   0.0  0:00.20 migration/7
   35 root        20   0       0       0       0  S   0.0   0.0  0:00.14 ksoftirqd/7
```

top

Извеждане на процесите под Windows

		0%	19%	1%	0%	1%	
		CPU	Memory	Disk	Network	GPU	GPU engine
Name		Status					
Apps (4)							
>	Microsoft PowerPoint		0%	186,1 MB	0 MB/s	0 Mbps	0%
>	SSH, Telnet and Rlogin client		0,1%	5,6 MB	0 MB/s	0 Mbps	0%
>	Task Manager		0,2%	23,2 MB	0,1 MB/s	0 Mbps	0%
>	Windows Explorer		0%	42,4 MB	0 MB/s	0 Mbps	0%
Background processes (80)							
>	64-bit Synaptics Pointing Enhancer		0%	2,1 MB	0 MB/s	0 Mbps	0%
>	Adobe Acrobat Update Service		0%	0,6 MB	0 MB/s	0 Mbps	0%
>	Antimalware Service Executable		0,1%	120,7 MB	0 MB/s	0 Mbps	0%
	Canon MF Network Scanner Sel...		0%	1,0 MB	0 MB/s	0 Mbps	0%
	CnTnrStsTask.exe (32 bit)		0%	1,0 MB	0 MB/s	0 Mbps	0%
	Component Package Support Se...		0%	1,1 MB	0 MB/s	0 Mbps	0%
>	Cortana (2)		0%	1,3 MB	0 MB/s	0 Mbps	0%

Въпроси?