Процеси

доц. д-р инж. Христо Вълчанов

http://cs.tu-varna.bg

Основни концепции

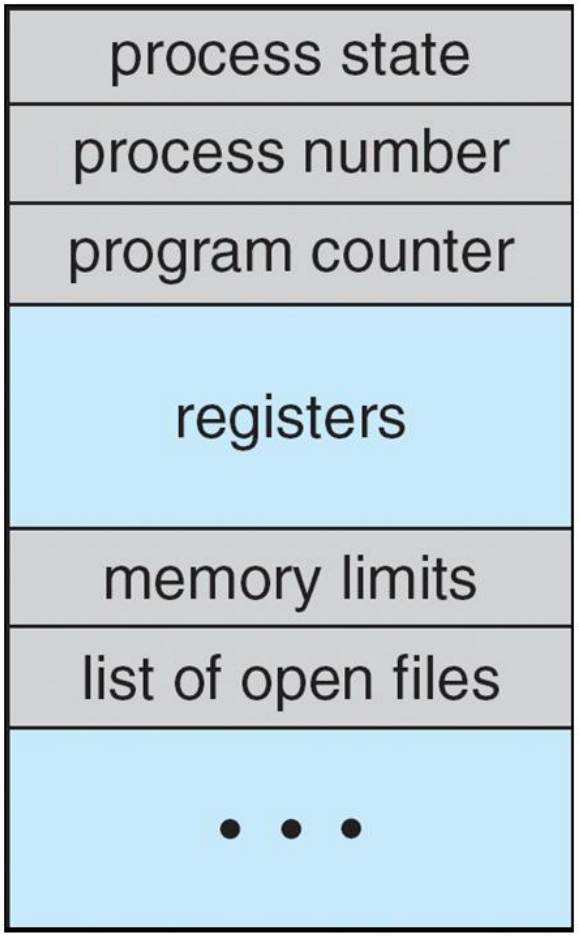
- **Процес** – програма по време на нейното изпълнение.
- Включва множество компонента:
  - Програмен код (text section).
  - Текущата му активност, включваща регистрите на процесора.
  - Стек.
  - Даннова част с глобални променливи.
  - Динамично заемана памет (Heap).

Програма и процес

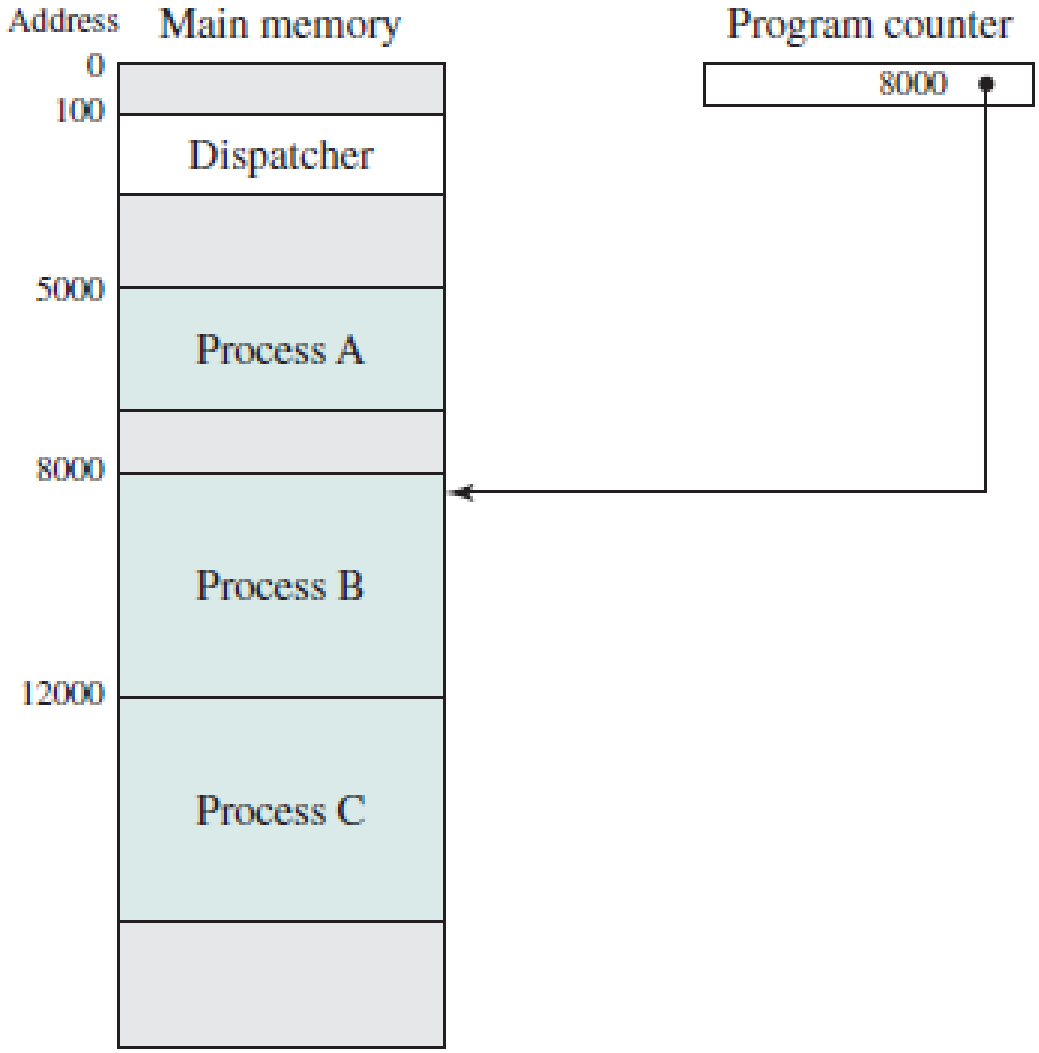
- **Програма** – пасивен компонент, съхранен на диска (изпълним файл).
- **Процес** – активен компонент.
- Програмата става процес, когато изпълнимият файл се зареди в паметта.
- Една програма може да се изпълнява като множество процеси.

Блок за управление на процеса

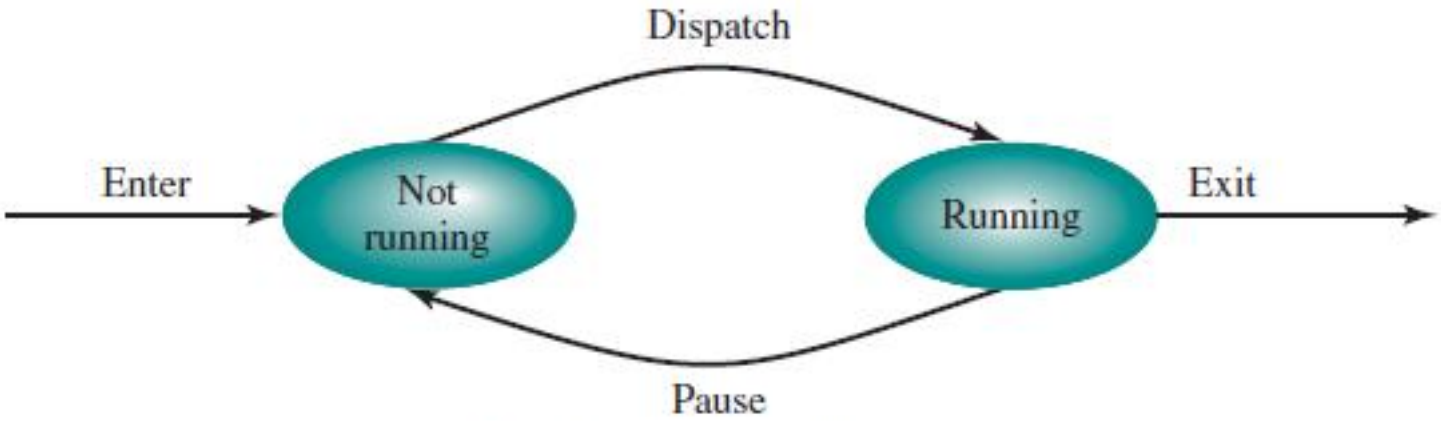
Информация, асоциирана с всеки процес (Process Control Block – PCB).

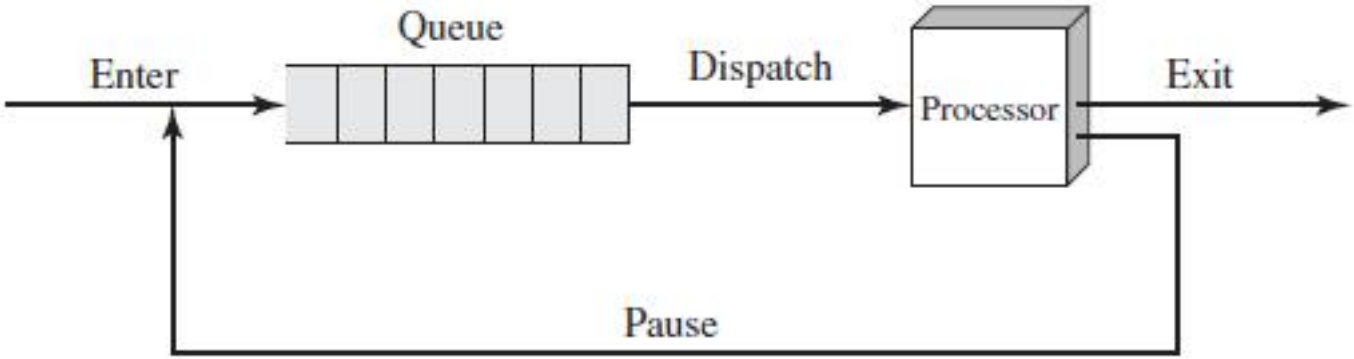


Процеси в паметта

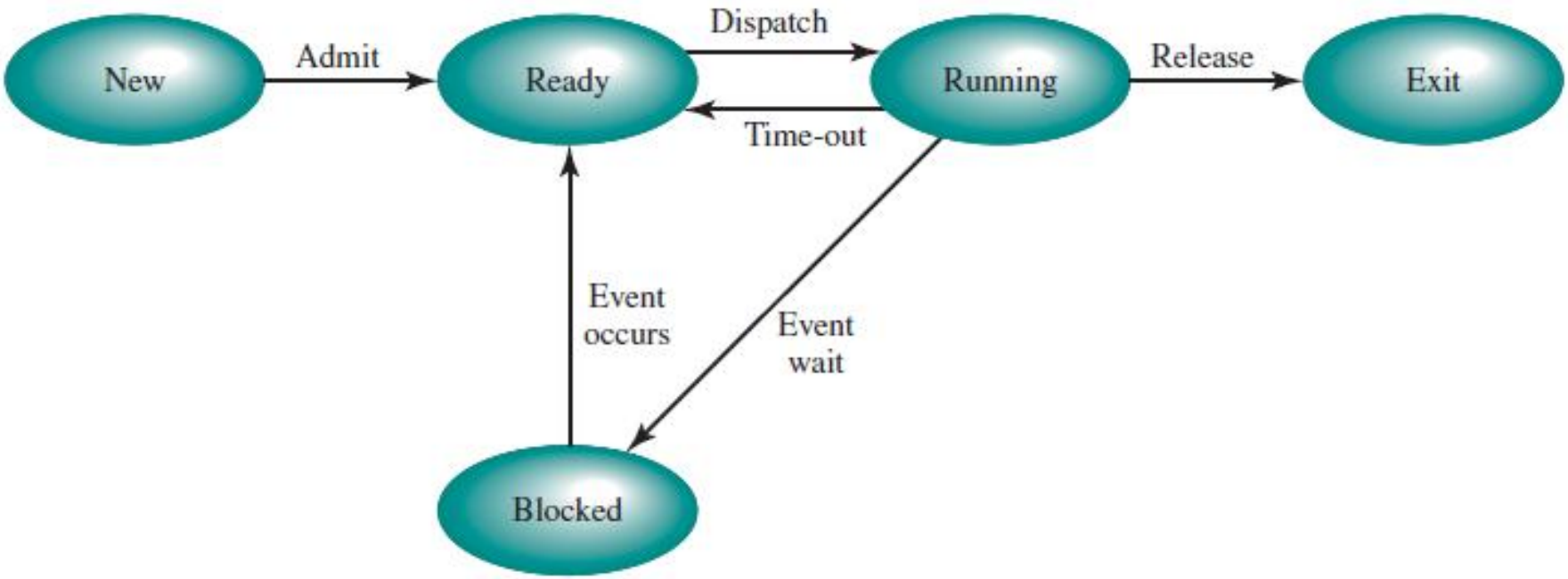


Модел с две състояния на процесите

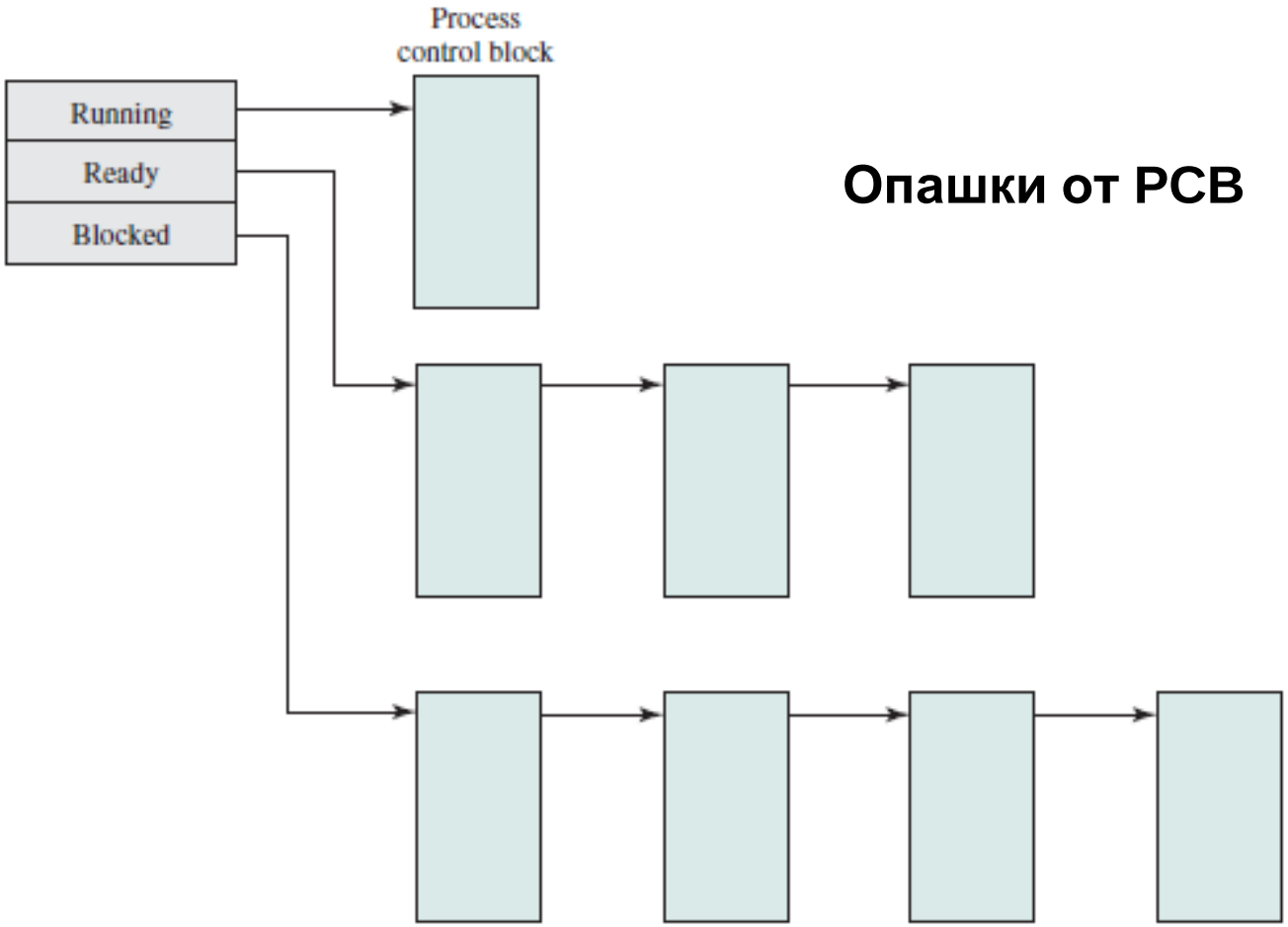




Модел с пет състояния на процесите

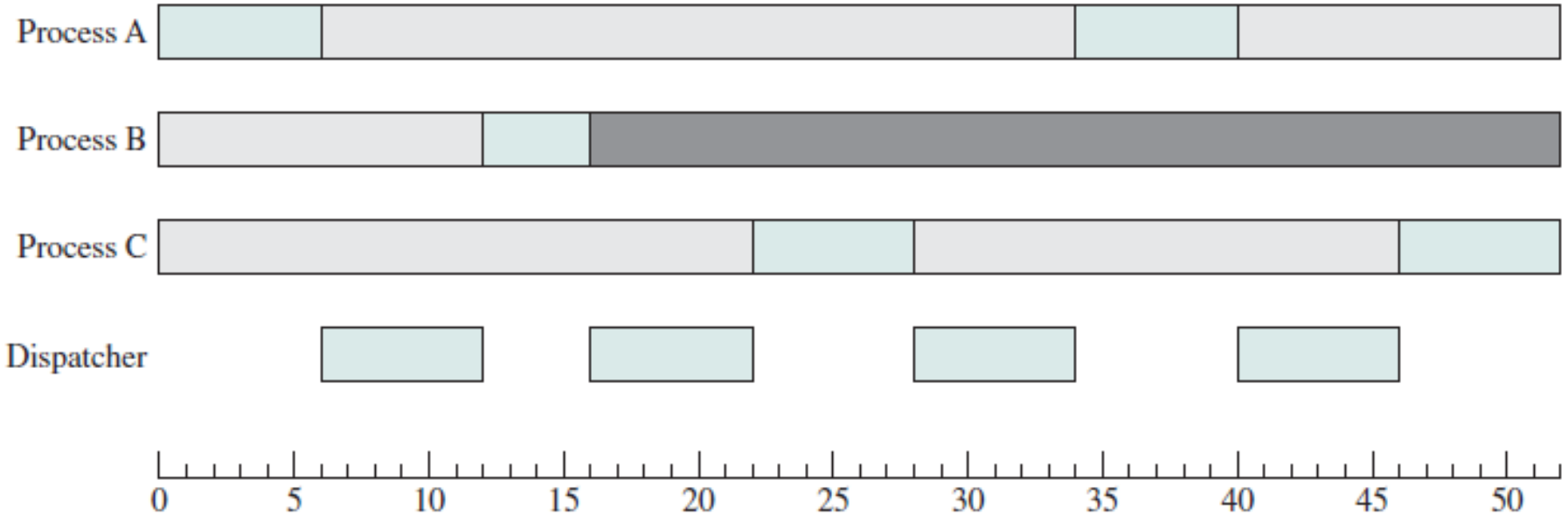


Организация на състоянията на процесите

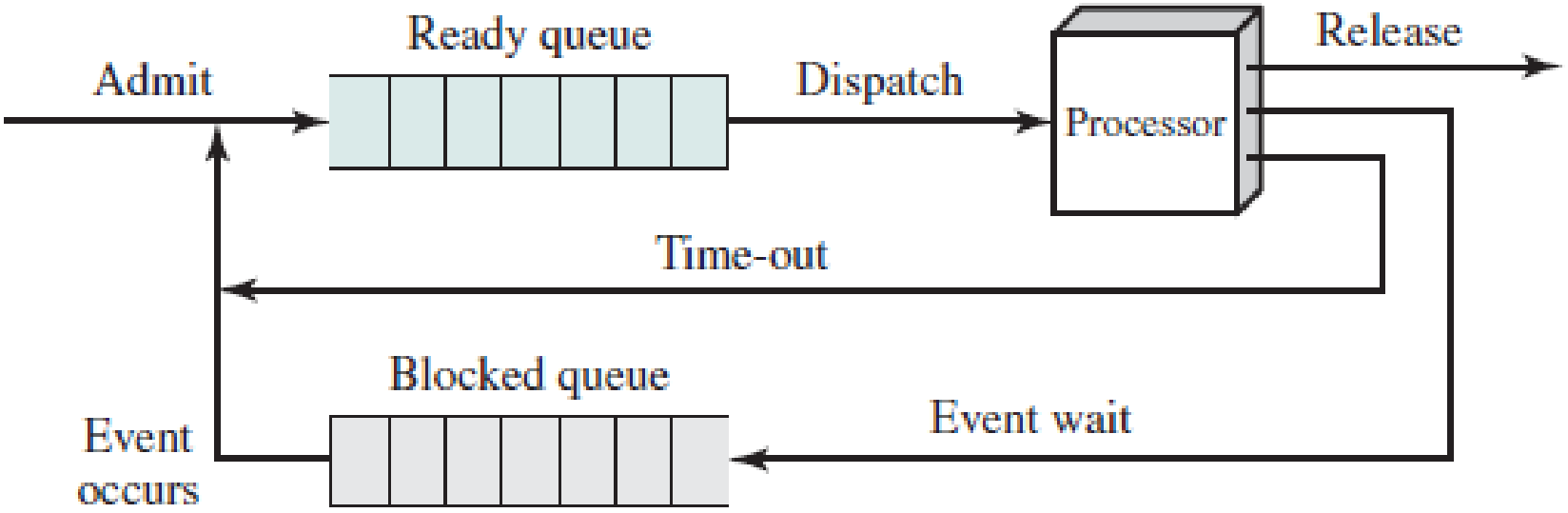


Опашки от PCB

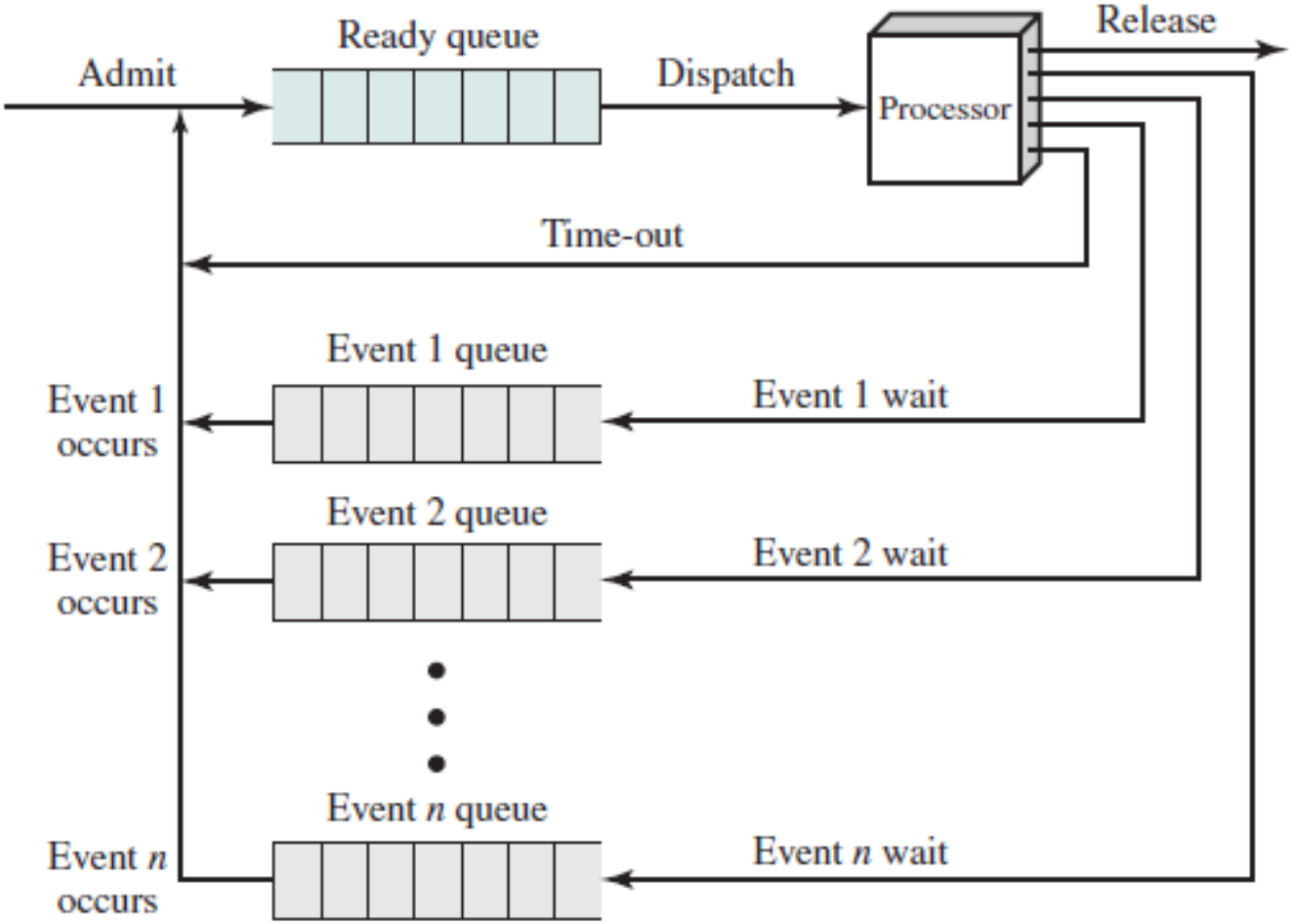
Изпълнение на процесите



Единична опашка на блокирани процеси



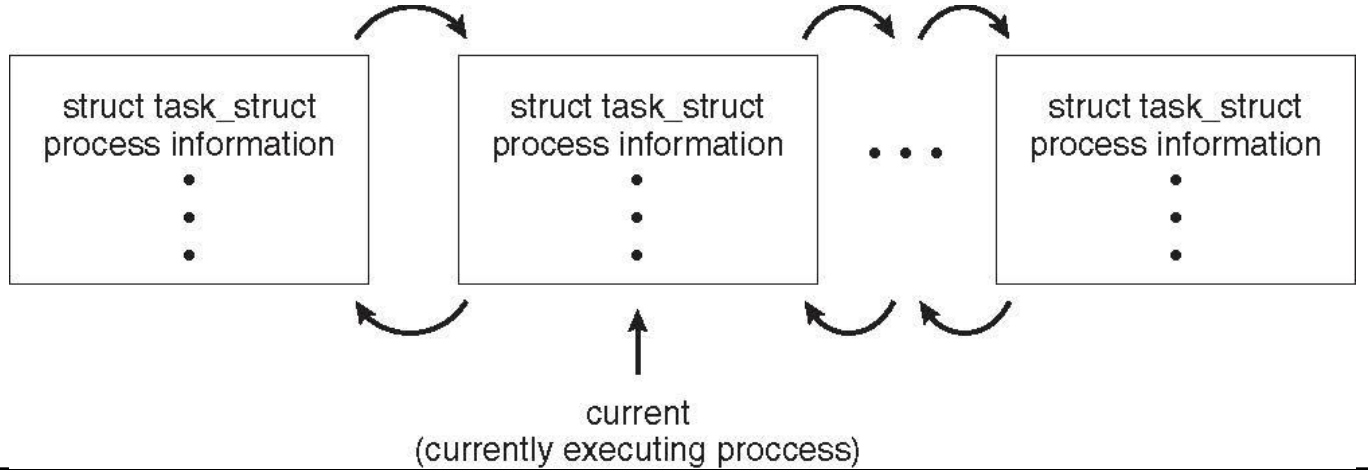
Множество опашки на блокирани процеси



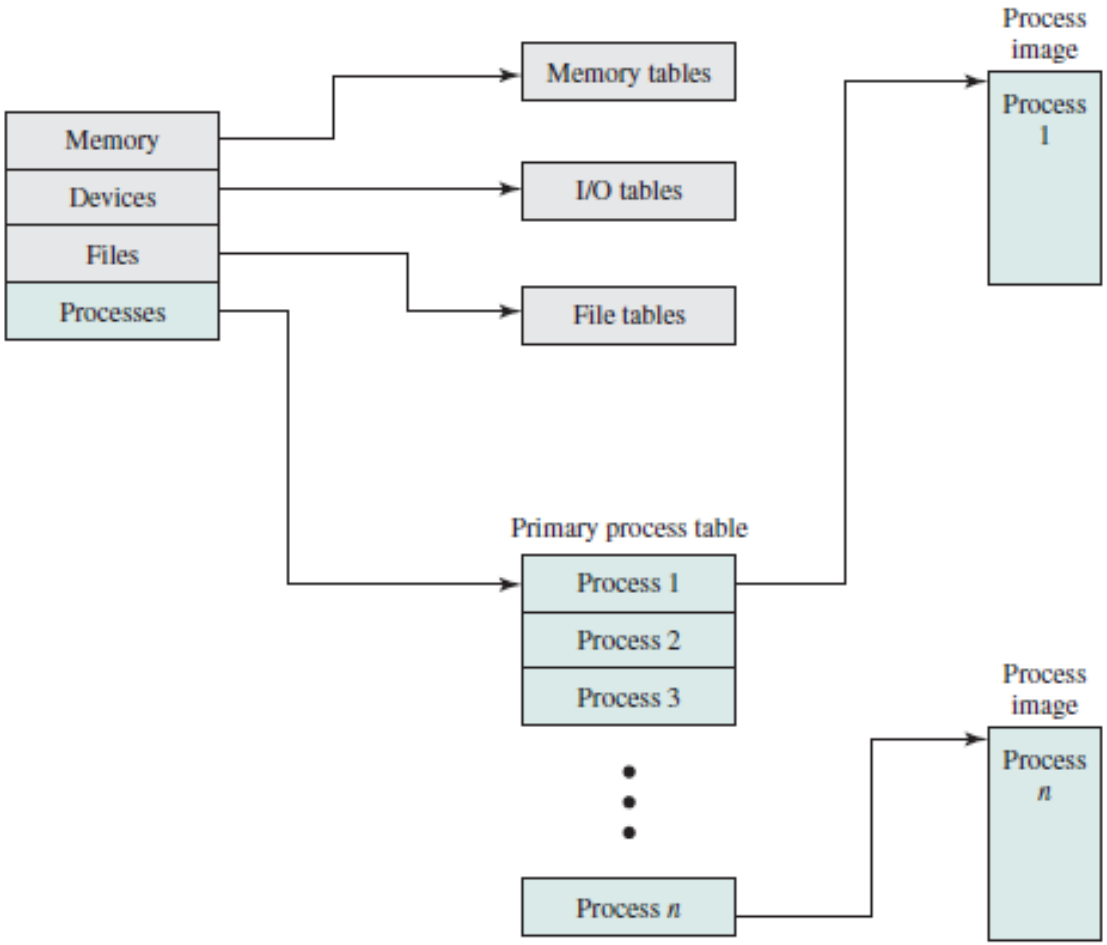
Реализация на процес в Linux

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 */
```



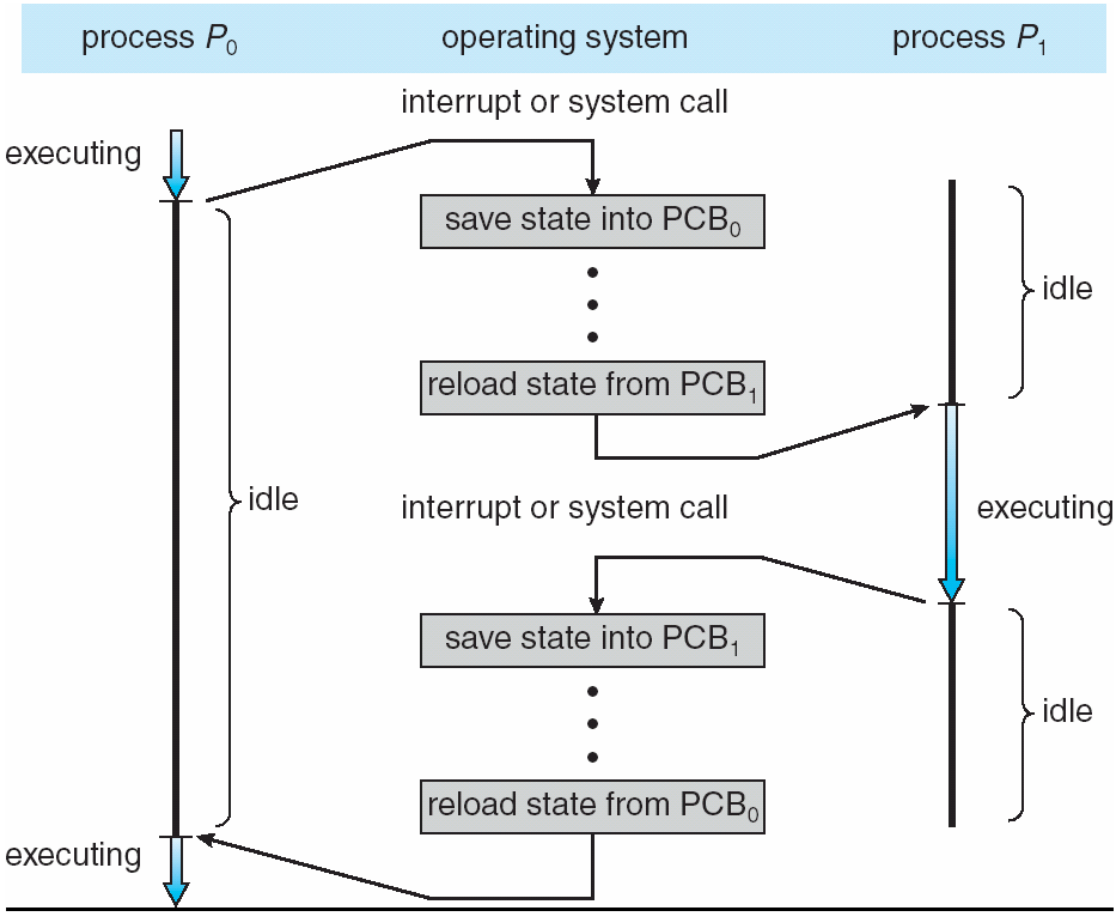
Структури данни на ОС за управление



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**).

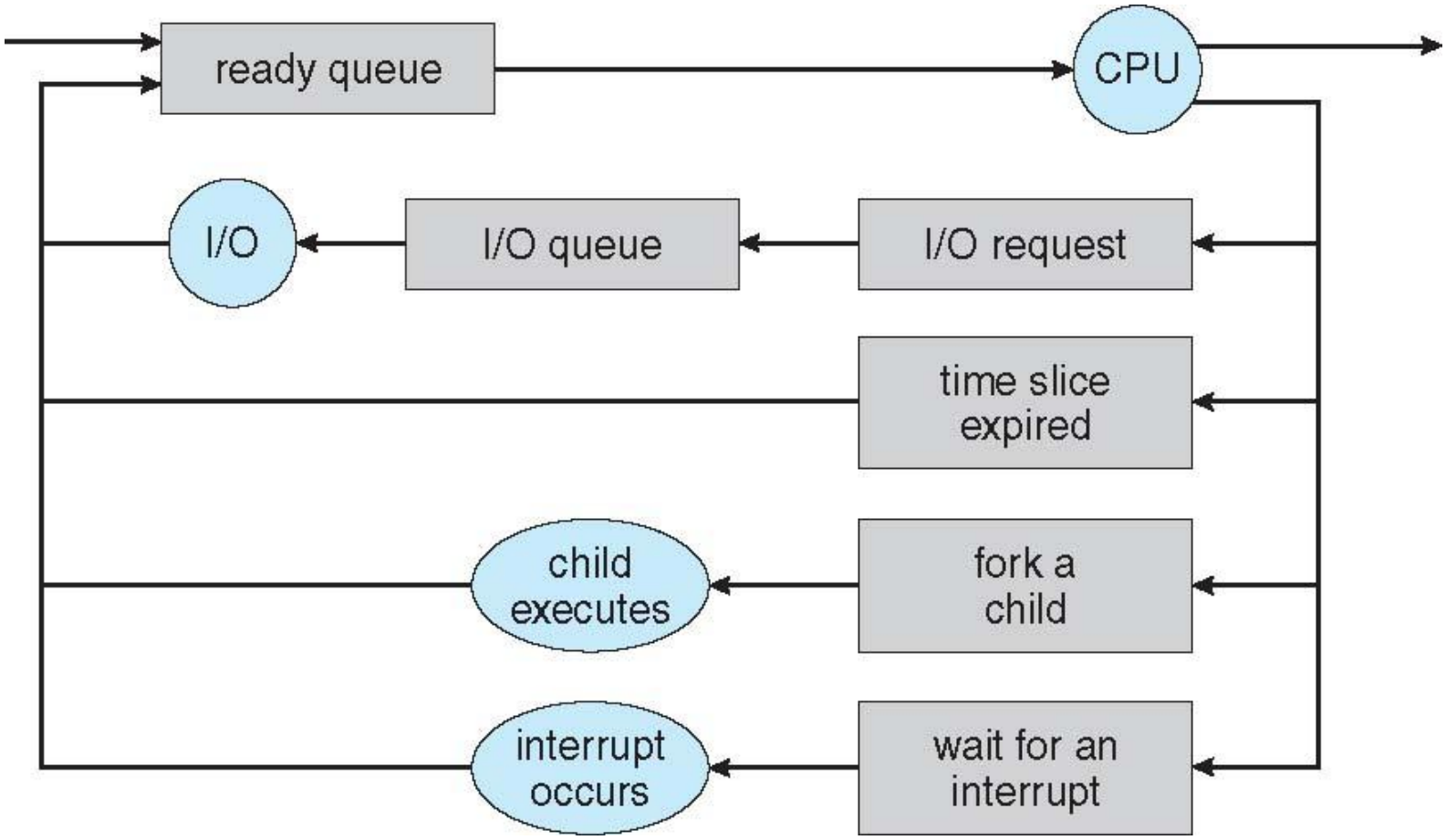
Превключване на процесора между процесите



Планиране на процесите

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

Диаграма на превключванията на процесите



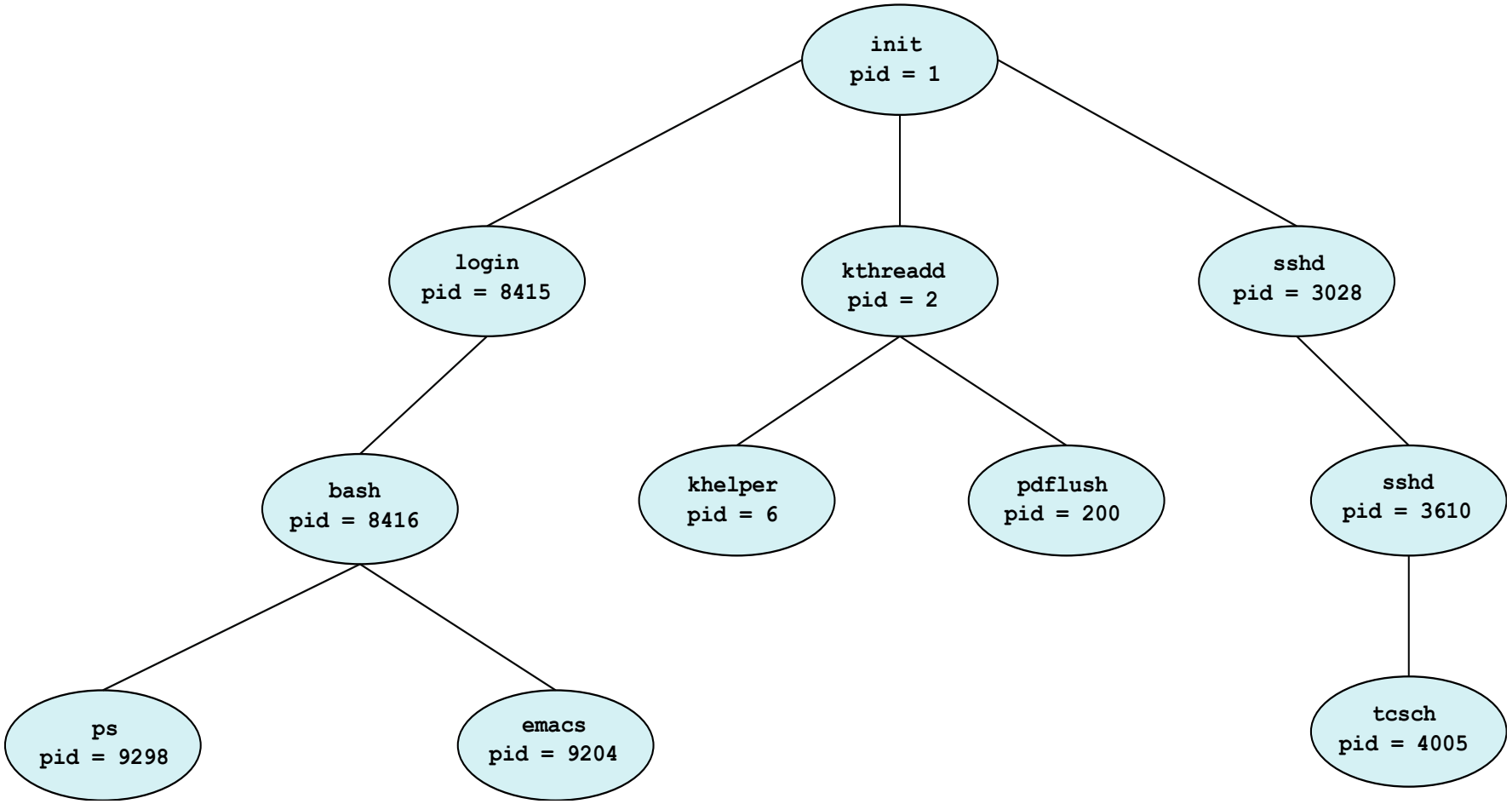
Основни операции върху процесите

- Създаване;
- Завършване.

Създаване на процес

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

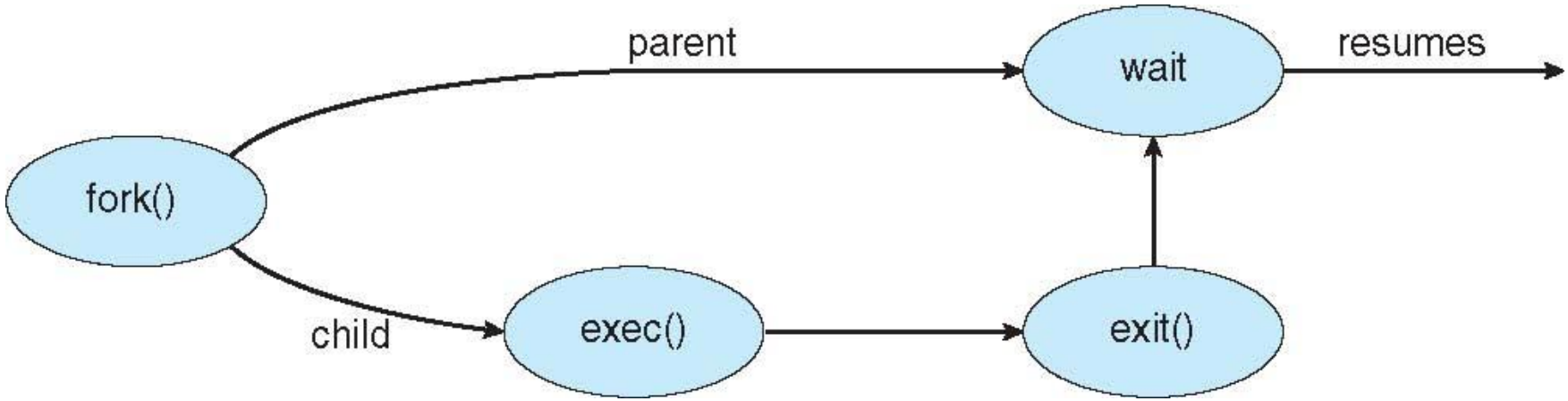
Дърво на процесите в Linux



Създаване на процес

- 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 - 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 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.

Въпроси?