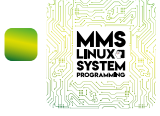
Системно програмиране за Линукс

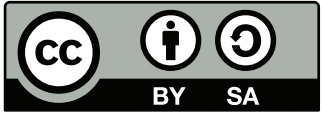
Управление на процесите. Дисциплини за планиране.

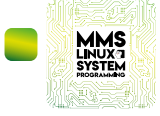
Ангел Чолаков

MMSolutions
A ThunderSoft Company

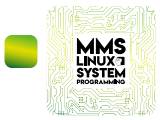
14.04.2021г.





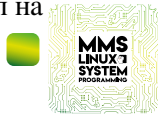


Съдържание I



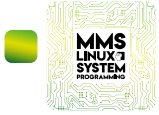
Цел на презентацията

- Да опита да поясни:
  - как ОС координира изпълнението на потребителски задания и на своите собствени процеси;
  - каква е ролята на системния диспечер;
  - какво е контролен блок за управление на процеси (PCB);
  - какви основни дисциплини за планиране съществуват;
  - очертае основни механизми за планиране на задачи в многоядрени или многопроцесорни системи;
  - кои са основните системни извиквания, обвързани с жизнения цикъл на заданията



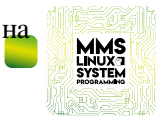
Въведение

- **Предизвикателства** при обслужването на програмни задания:
  - как процесорът или процесорните ядра да бъдат натоварени оптимално;
  - как да се координира обработването на входно-изходни операции;
  - как да се планира степента на натовареност с оглед на обема постъпващи задачи;
  - как нивото на мултипрограмиране да се поддържа стабилно във времето;
  - как се обезпечава обслужване на множество потребители и/или заявки във времето



ОС в ролята на мениджър на системни ресурси

- Основни ангажименти:
  - позволи изпълнение на множество потребителски приложения в прогнозируемо време;
  - да **гарантира отзивчивост** на системата, като намали възможността за нежелан срив;
  - да **планира системното натоварване** и осигури балансиран поток от обслужвани задания;
  - да **определя адекватно избора на процеси за изпълнение** съобразно потребностите и заложените цели;
  - да предостави възможност за управление и конфигуриране степента на натовареност

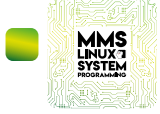


Какво е многозадачност?

Дефиниция:

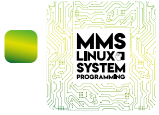
- възможност за последователно редуващо се във времето или паралелно изпълнение на множество потоци от инструкции, които идентифицират едно или повече различни задания

Как се подсигурява среда за многозадачност при еднопроцесорни системи?



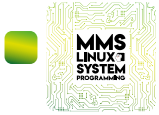
Кооперативен подход

- **Същност**:
  - ОС се доверява на разработчиците на приложения;
  - ОС делегира отговорността на автора на заданието да поеме контрол върху системните ресурси и да ги освободи своевременно, когато е готов
- **Контролът се връща на ОС при**:
  - нормално завършване на задачата;
  - заявка за изпълняваща се по-бавна входно-изходна операция;
  - възникване на изключителна ситуация - грешка

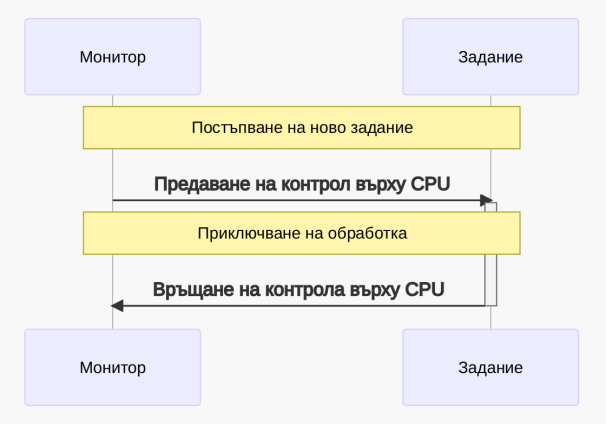


Системни извиквания

- Разновидности:
  - тип **trap** - инициира превключване на контекст;
  - тип **yield** - сигнализира заявка за освобождаване на заетия изчислителен ресурс;
  - тип **interrupt** - свързани с обслужване на събитие, маркирано от хардуерно прекъсване;



Кооперативен подход: системна координация

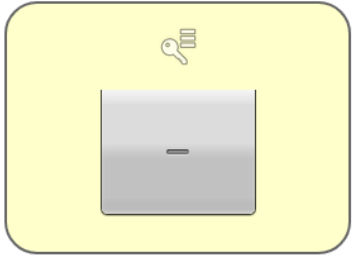


pic. based on `https://en.wikipedia.org/wiki/Cooperative_multitasking`

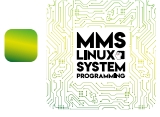
Кооперативен подход: недостатъци

Открояват се:

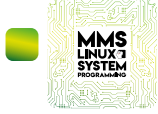
- липса на механизъм за детекция и справяне с логически дефекти;
- липса на защитни инструменти за предотвратяване на последици от злонамерен софтуер, който монополизира процесора;
- всеки генерален пропуск за навременно връщане на управлението на ОС води до компрометиране функционалността на системата



pic. own work

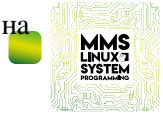


Какво решение се предлага на така описаните проблеми?



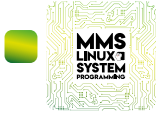
Подход с времеделене

- Същност:
  - отговорността по планиране на постъпващите задачи се поема изцяло от ОС;
  - разчита се на просто хардуерно осигуряване - **програмируем таймер** и **механизъм на генериране и обработка на прекъсвания**;
  - задачите се обслужват на порции посредством **регулярно периодично редуване** на изпълняваните процеси до тяхното завършване;
  - интервалите на обслужване и превключването на контекста се управляват с помощта на таймерните прекъсвания;
  - ОС разполага с привилегирован достъп до таймера и обработването на прекъсванията

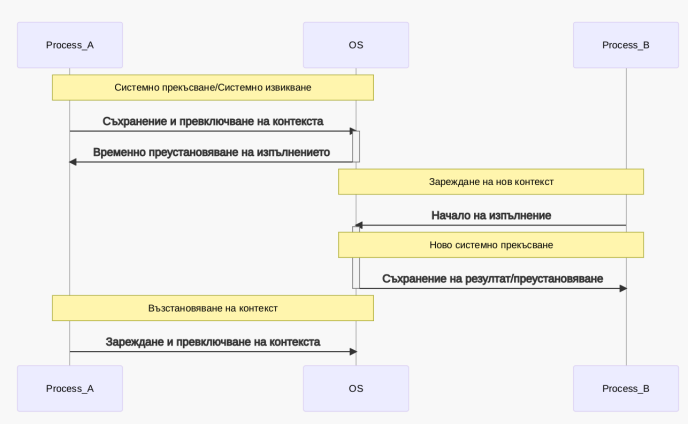


Подход с времеделене

- Отговорности на ОС:
  - **проследяване** състоянието на изпълнение на наличните задачи;
  - **съхранение на локалните данни и регистрови структури** на текущо изпълнявания процес преди превключването;
  - наличие на специализирани подпрограми за **обслужване на апаратно прекъсване** и манипулиране на контекста;
  - **възстановяване** на вече съхранените локални данни и регистрови структури преди възобновяване на прекъснат процес



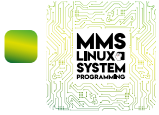
Да си припомним процедурата по превключване на контекст



pic. based on `https://web.archive.org/web/20100218115342/http://www.linfo.org/context_switch.html`

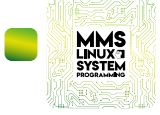
Подход с времеделене

- **Предизвикателства**:
  - апаратната поддръжка преодолява проблема, свързан с нежелано заемане на процесора от една програма;
  - остава обаче необходимост от проследяване характера на работното натоварване и планирането на заданията;
  - никой не желае важна задача да бъде прекъсвана и отлагана постоянно

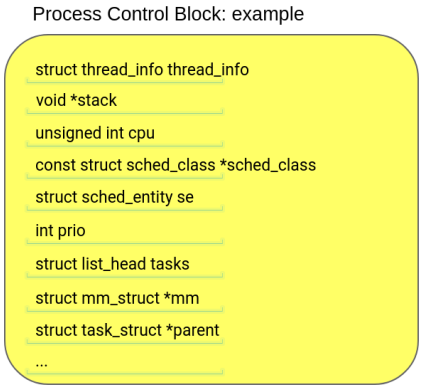


Контролен блок за управление на процеси

- Предназначение:
  - описва **системни атрибути**, които представят изпълнявания процес и подпомагат работата на подсистемите за планиране на задачи;
  - използва се за представяне на **контекста** на изчисление;
  - участва в механизмите за превключване на процесора и предаване управлението на друг процес



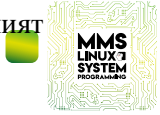
Диаграма на управляващ блок на процес



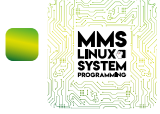
pic. based on `https://elixir.bootlin.com/linux/latest/source/include/linux/sched.h`

Структура на управляващ блок на процеса

- Отразява обикновено:
  - състоянието на процеса: чакащ изпълнение, изпълняващ се, блокиран, терминиран и т.н.;
  - съдържанието на програмния брояч, указващ поредната инструкция за изпълнение;
  - съдържание на програмния стек, което отразява етапа на изпълнение на задачата;
  - регистрови структури на процесора, които са ангажирани: PC, LR or FP, акумулатори, флагове и др.;
  - данни за сегментите виртуална памет, в които се помества изпълнимият образ и др.

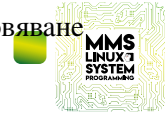


А как е огранизирано планирането на процесите?



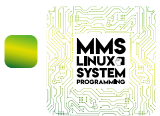
Системен диспечер на заданията

- **Роля**:
  - **управлява работното натоварване**, отразено чрез количеството постъпващи задания;
  - **имплементира политики за планиране** на процесите на високо логическо ниво;
  - извършва **мониторинг** на зададени критерии за оценка на производителността;
  - предоставя **механизми за обработка на системни прекъсвания** и превключване на процесора на ниско ниво;
  - възползва се от механизми за манипулиране, съхранение и възстановяване на контекста

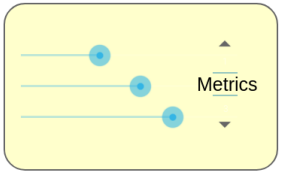


Диспечер на заданията

- Критерии за производителност:
  - **абсолютна степен на заетост** на системните ресурси (количесво разходвана памет, процесорни ядра, набор ангажирани входно-изходни устройства);
  - **относително време за изчакване** в опашката от постъпващи задания спрямо набора задачи;
  - **относително време за изпълнение** спрямо количеството на задачите и характера на обработките



Традиционни метрики



pic. own work

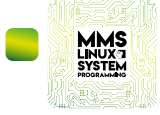
Примери:

- време за изпълнение на задание: **T turnaround**;
- средно време за чакане;
- средно време за отговор или резултат: **T response**

Дефиниция на най-често срещаните метрики:

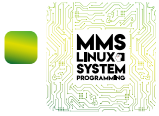
$$T_{turnaround} = T_{completion} - T_{arrival} \quad (1)$$

$$T_{response} = T_{firstrun} - T_{arrival} \quad (2)$$

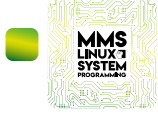


Начални допускания при част от разглежданите примери

- Приема се:
  - фиксиран характер на работния товар и фиксиран брой процеси;
  - всички заявки постъпват първоначално в приблизително еднакъв момент от време;
  - не съществува задача с неизвестно прогнозирано време за обработка;
  - визират се разновидности планиране с изместване (preemptive)

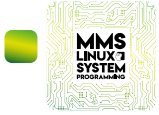


А какви са възможните политики за планиране?

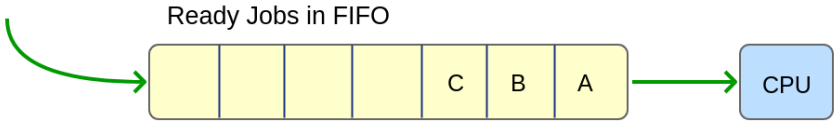


Дисциплина FIFO: първи дошъл - първи обслужен

- **Същност**:
  - илюстрира общоразпространен принцип на подредба на заданията в единична опашка;
  - обслужването става според реда на постъпване



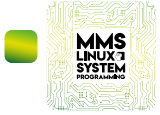
Дисциплина FIFO: диаграма



pic. based on `https://en.wikipedia.org/wiki/Scheduling_(computing)`

Дисциплина FIFO: разновидности

- Някои от най-важните са:
  - ”**първи дошъл - първи обслужен (FCFS)**” - без изместване на активна задача;
  - ”**най-краткото задание първо (SJF)**” - да премахнем ли изискването задачите да постъпват в един момент от време?;
  - ”**заданието с най-малко оставащо време до завършване първо (STCF)**” - тук би трябвало да премахнем и изискването всяка задача да се изпълнява напълно от момента на постъпване



FIFO: сравнителна илюстрация

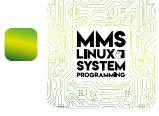
FIFO	time, s:	140	160	180	200	220	240	Avg. Turnaround time,s:
	job, №	1	2	3				160.00

FIFO (SJF)	time, s:	20	40	180	200	220	240	Avg. Turnaround time,s:
	job, №	1	2	3				80.00

t=20:(2,3 arrive)

FIFO (STCF)	time, s:	20	40	60	180	200	220	Avg. Turnaround time,s:
	job, №	1	2	3	1			80.00

pic. based on `https://pages.cs.wisc.edu/~remzi/OSTEP/cpu-sched.pdf`

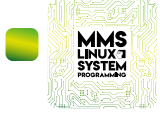


FIFO: пояснение на метриките в илюстрацията

$$T_{turnaround(FIFO)} = \frac{(140 + 160 + 180)}{3} \tag{3}$$

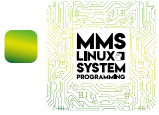
$$T_{turnaround(SJF)} = \frac{(20 + 40 + 180)}{3} \tag{4}$$

$$T_{turnaround(STCF)} = \frac{((180 - 0) + (40 - 20) + (60 - 20))}{3} \tag{5}$$

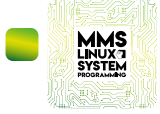


Дисциплина RR: кръгова или циклична

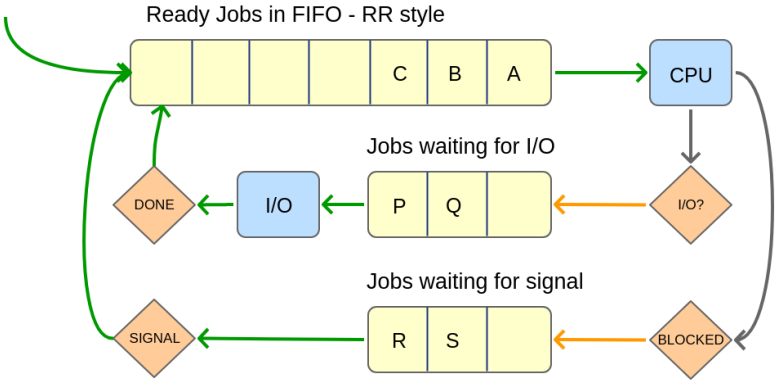
- **Същност**:
  - стои в основата на интерактивните системи с времеделене;
  - всяка задача постъпва в опашка, като получава процесора за фиксиран квант от време и се нарежда обратно в опашката до приключване



Как би могла да се имплементира RR политика на практика?



Диаграма на RR имплементация с няколко опашки



pic. based on `https://pages.cs.wisc.edu/~remzi/OSTEP/cpu-sched.pdf`

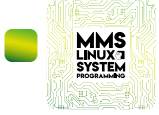
Дисциплина RR: илюстрация

RR	time, s:	30			60			**Avg. Turnaround time,s:**
	t_quant, s:	10	10	10	10	10	10	
	job, №	1	2	3	1	2	3	50.00

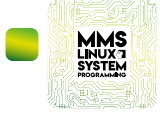
pic. based on `https://pages.cs.wisc.edu/~remzi/OSTEP/cpu-sched.pdf`

Критерий справедливост на планиране

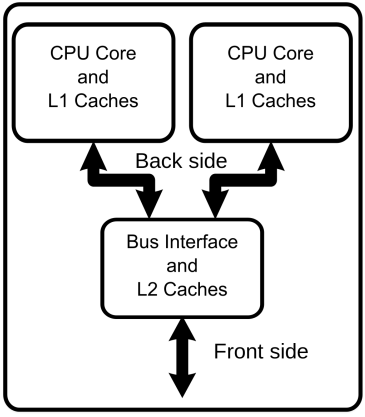
- **Дефиниция**:
  - базова оценка дали планирането е справедливо (**fair**) по отношение в каква степен алгоритъмът е склонен да обработва един клас задачи за сметка на друг



Какви решения съществуват при SMP системите?



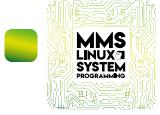
Диаграма на SMP система



pic. by CountingPine, Public Domain via Wikimedia Commons

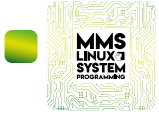
Планиране при многопроцесорни/многоядрени системи

- Подходи:
  - вземат под внимание механизмите за работа с **кеш паметта**;
  - състоянието на обновяваните зони в кеша и **инвалидирането** им изисква нови инструменти за синхронизация

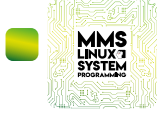


Дисциплини при SMP системи

- Примери:
  - **SQMS** - с една споделена опашка и миграция на задачи;
  - **MQMS** - с отделна обособена опашка за всеки процесор

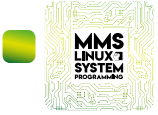


Как се диспечерират процесите в Линукс?



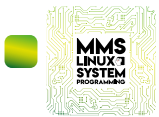
Линукс диспечер: кратка предистория

- Характеристики на O(1) алгоритъм:
  - нуждае се от **константно време** за избор на задача независимо от броя процеси;
  - разновидност на RR с времеделене и фиксиран квант от време;
  - поддържат се две опашки - на активни и неактивни процеси;
  - избраният от списъка с активни процес се изпълнява за времевия квант и постъпва в опашка на неактивни (изчакващи) задания;
  - при изчерпване на готовите за изпълнение процеси - ролята на двете опашки се сменя



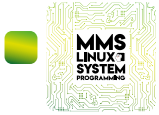
Линукс диспечер: CFS

- Характеристики на CFS диспечер:
  - въвежда различни **класове на планиране**, в които процесите се групират;
  - всеки клас се характеризира с различен **приоритет и алгоритъм за обслужване**;
  - пропорция от процесорното време се предоставя според степента на кооперативност на един процес (**nice value**);
  - не се възползва от установени времеви отрязъци, а идентифицира средното време, което един процес прекарва в очакване да получи обслужване;
  - с нарастване на времето за изчакване на получаване на обслужване, нараства и относителният приоритет на процеса



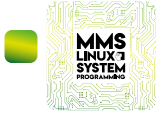
Степен на кооперативност: (nice value)

- Обхват от стойности:
  - от -20 до +19 десетично;
  - по-ниска стойност указва по-висок приоритет;
  - промяна на приоритет може да бъде заявена чрез **renice** например;
  - Линукс поддържа и дисциплини за работа в реално време: **SCHED_FIFO** и **SCHED_RR**;
  - действителният обхват от приоритети е разделен на два подинтервала: [0-99] за реалновремеви и [100-139] за такива с нормален приоритет



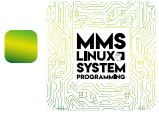
Разкриване на йерархията на процесите

```
$ ps -e --forest

  2292 ?        00:00:09  \_ gvfs-udisks2-vo
  2297 ?        00:00:00  \_ gvfs-goa-volume
  2301 ?        00:00:00  \_ gvfs-afc-volume
  2306 ?        00:00:00  \_ gvfs-mtp-volume
  2310 ?        00:00:00  \_ gvfs-gphoto2-vo
  2328 ?        00:00:01  \_ ibus-x11
  2330 ?        00:00:01  \_ ibus-portal
  2400 ?        00:09:45  \_ teams
  2410 ?        00:00:00  |   \_ teams
  2619 ?        00:00:10  |   |   \_ teams
  2620 ?        00:00:00  |   |       \_ teams
  2468 ?        00:19:12  |   \_ teams
  2471 ?        00:00:00  |   |   \_ teams
  2469 ?        00:02:37  |   \_ teams
  2497 ?        00:00:01  |   \_ teams
  2569 ?        00:49:27  |   \_ teams
  2636 ?        02:02:00  |   \_ teams
 16899 ?        00:00:05  |   \_ teams
```



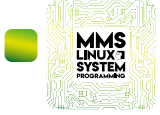
Мониторинг на приоритетитие на процесите

```
$ ps -e -o pid,ppid,pri,cmd

    0    7087       2   39 [kworker/u17:7]
    0    7420       2   19 [kworker/0:0]
    0    7823       2   39 [kworker/u17:0]
    0    8032       2   19 [kworker/6:0]
    0    8193       2   19 [kworker/3:2]
    0    8388       2   19 [kworker/3:1]
    0    8419       2   19 [kworker/4:1]
    0    8800       2   19 [kworker/u16:1]
    0    8989       2   19 [kworker/u16:4]
    0    9007       2   19 [kworker/4:0]
    0    9145       2   39 [kworker/u17:4]
    0    9407       2   19 [kworker/7:1]
    0    9563       2   19 [kworker/u16:2]
    0    9564       2   19 [kworker/u16:3]
```

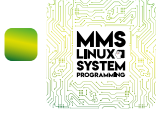


Състояния на процесите в Линукс

```
D    uninterruptible sleep (usually IO)
R    running or runnable (on run queue)
S    interruptible sleep (waiting for an event to complete)
T    stopped by job control signal
t    stopped by debugger during the tracing
W    paging (not valid since the 2.6.xx kernel)
X    dead (should never be seen)
Z    defunct ("zombie") process, terminated but not reaped by its parent
```

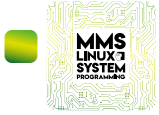


Практическа част: демонстрация на fork, wait, exit и exec системни извиквания

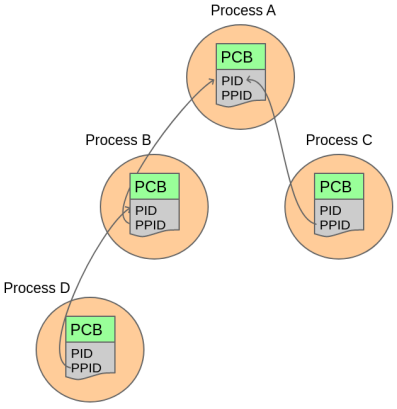


Fork системно извикване

- Предназначение:
  - създава нов подчинен процес, наречен **потомък (child)**
- Потомъкът се изпълнява в копие на виртуалното адресно пространство на родителския процес, като:
  - новият процес получава свой **уникален идентификатор (PID)**;
  - новият процес не наследява семафорни примитиви за манипулация на адресното пространсто, както и структури, дефиниращи операции за работа със семафори;
  - новият процес наследява обаче копия на файловите дескриптори, принадлежащи на родителя



Диаграма на йерархията между процесите

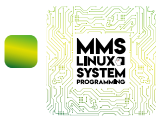


pic. based on `https://en.wikipedia.org/wiki/Pstree`

Fork примерна програма: начало

//C

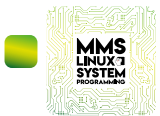
```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/wait.h>
#include "test.h"

int main(int argc, char *argv[]) {

        int pid, ret_status;

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



Fork примерна програма: продължение

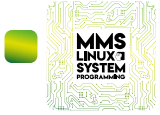
//C

```
        /* Check for child process - pid == 0 */
        if (!pid) {
                sleep(1);
                fprintf(stdout, "child runs with pid: %d and parent pid: %d\n",
                        getpid(), getppid());
        } else {
                fprintf(stdout, "parent runs with pid: %d and parent pid: %d\n",
                        getpid(), getppid());
                wait(0);
                exit(EXIT_SUCCESS);
        }
}
```



Wait системно извикване

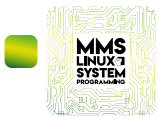
- Предназначение:
  - временно преустановява изпълнението на извикващия процес до получаване на нотификация за промяна в състоянието на процеса потомък
- Особености:
  - промяна в състоянието на процеса потомък се указва чрез: терминиране, прихващане на сигнал за стопиране или възобновяване;
  - в случая на терминиран процес, **wait** позволява на родителя да освободи ресурсите, асоциирани с процеса потомък;
  - ако родителският процес не се възползва от услугите на wait, то терминираният процес наследник остава в ”**зомби**” състояние



Осиротял процес потомък: пример

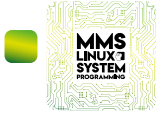
//C

```
        /* Check for child process - pid == 0 */
        if (!pid) {
                sleep(5);
                fprintf(stdout, "child runs with pid: %d and parent pid: %d\n",
                        getpid(), getppid());
        } else {
                fprintf(stdout, "parent runs with pid: %d and parent pid: %d\n",
                        getpid(), getppid());
                exit(EXIT_SUCCESS);
        }
}
```



Exec фамилия от системни извиквания

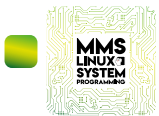
- Предназначение:
  - подменя съдържанието на виртуалното адресно пространство, заделено за процес наследник, с изображение на нов външен процес, идентифициран чрез име на програма и съпътстващи аргументи



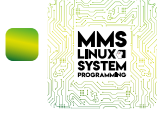
Модификация на образа на процес наследник: пример

//C

```
        /* Check for child process - pid == 0 */
        if (!pid) {
                fprintf(stdout, "child execs ls -al");
                ret_status = execlp("ls", "ls", "-al", "./", (char *)NULL);
                fprintf(stdout, "Error running execlp ret: %d\n",
                        ret_status);
        } else {
                fprintf(stdout, "parent runs with pid: %d and parent pid: %d\n",
                        getpid(), getppid());
                waitpid(pid, NULL, 0);
                exit(EXIT_SUCCESS);
        }
}
```

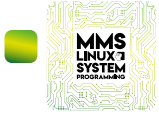


Това не е всичко! Има много повече възможности и примери, описани в man pages...



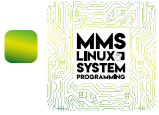
Бележки по материалите и изложението

- материалът е изготвен с образователна цел;
- съставителите не носят отговорност относно употребата и евентуални последствия;
- съставителите се стремят да използват публично достъпни източници на информация и разчитат на достоверността и статута на прилаганите или реферирани материали;
- текстът може да съдържа наименования на корпорации, продукти и/или графични изображения (изобразяващи продукти), които може да са търговска марка или предмет на авторско право - ексклузивна собственост на съотнесените лица;
- референциите могат да бъдат обект на други лицензи и лицензни ограничения;
- съставителите не претендират за пълнота, определено ниво на качество и конкретна пригодност на изложението;
- съставителите не носят отговорност и за допуснати фактологически или други неточности;
- свободни сте да създавате и разпространявате копия съгласно посочения лиценз;



Референции към полезни източници на информация

- https://en.wikipedia.org/wiki/Process_management_(computing)
- https://en.wikipedia.org/wiki/Scheduling_(computing)
- https://en.wikipedia.org/wiki/Shortest_job_next
- https://pages.cs.wisc.edu/~remzi/OSTEP/cpu-sched-multi.pdf
- https://en.wikipedia.org/wiki/Cooperative_multitasking
- https://www.linuxjournal.com/node/10267
- https://man7.org/linux/man-pages/man2/fork.2.html
- https://man7.org/linux/man-pages/man2/waitid.2.html
- https://linux.die.net/man/3/execl
- https://en.wikipedia.org/
- https://search.creativecommons.org/



Благодаря Ви за вниманието!