Процес
(*Rev: 1.5*)

Любомир Чорбаджиев[1]
lchorbadjiev@elsys-bg.org

[1]Технологическо училище "Електронни системи"
Технически университет, София

23 октомври 2007 г.

Съдържание

Въведение

- Компютърната система се състои от набор от хардуерни ресурси.
- Софтуерните приложения се разработват за да изпълнят някаква задача.
- Неефективно е приложенията да се разработват директно за дадена хардуерна платформа.
- Операционната система предоставя удобен за използване, богат на възможности и еднообразен интерфейс, който може да се използва от приложенията.
- Операционната система предоставя еднообразно абстрактно представяне на ресурсите, които могат да се използват от едно приложение.

Изпълнение на приложения

- Изпълнението на приложенията се управлява от операционната система.
- Операционната система разпределя ресурсите между приложенията, които се изпълняват.
- Процесорът се превключва между приложенията, които се изпълняват.
- Това позволява по-ефективно използване на процесора и входно/изходните устройства.
- Операционните системи изпълняват различни приложения:
  - Системите за пакетна обработка – задания.
  - Системите с времеделене – потребителски програми или задачи.
- Задание, задача, процес <=> процес.

Дефиниция за процес

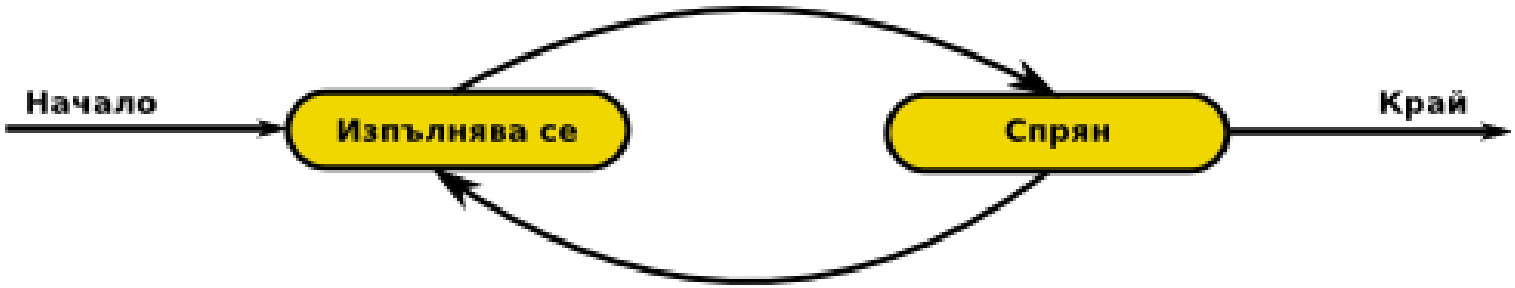
- Процесът е програма, която се изпълнява.
- Инструкциите на програмата се изпълняват последователно.
- Единица работа, която може да се изпълнява върху процесора.
- Единица работа, която се характеризира с изпълнение на последователност от инструкции, текущо състояние и множество от системни инструкции.
- Операционната система управлява изпълнението на процесите:
  - Изпълнява няколко процеса за да подобри използването на процесора.
  - Заделя ресурси, необходимо за изпълнението на процесите.
  - Предоставя механизми за комуникация между процесите.
  - Предоставя средства за създаване на процеси от потребителя.

Контролен блок на процеса

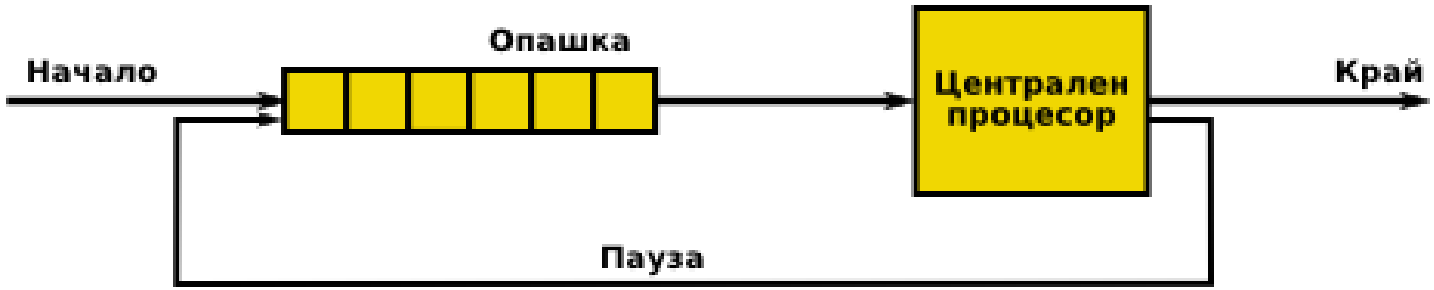
- Съдържа информацията, необходима на операционната система за управление на процеса:
  - Идентификация на процеса.
  - Състояние на процеса.
  - Приоритет.
  - Програмен брояч.
  - Данни за контекста на процеса.
  - Информация за управление на паметта.
  - Информация за състоянието на входно/изходните операции.
- Създава се и се управлява от операционната система.
- Позволява на операционната система да поддържа много процеси.

Модел на процеса с две състояния

- Всеки процес може да бъде само в две състояния:
  - Изпълнява се.
  - Не се изпълнява.



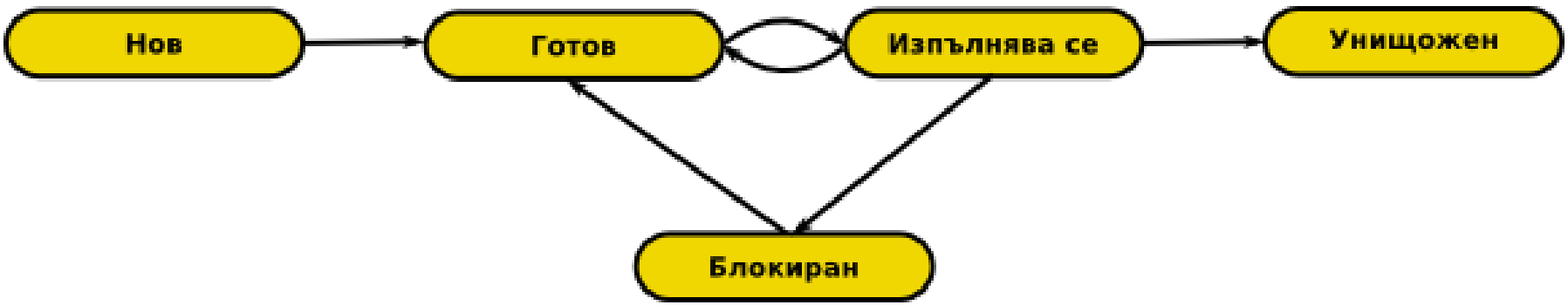
Модел на процеса с две състояния



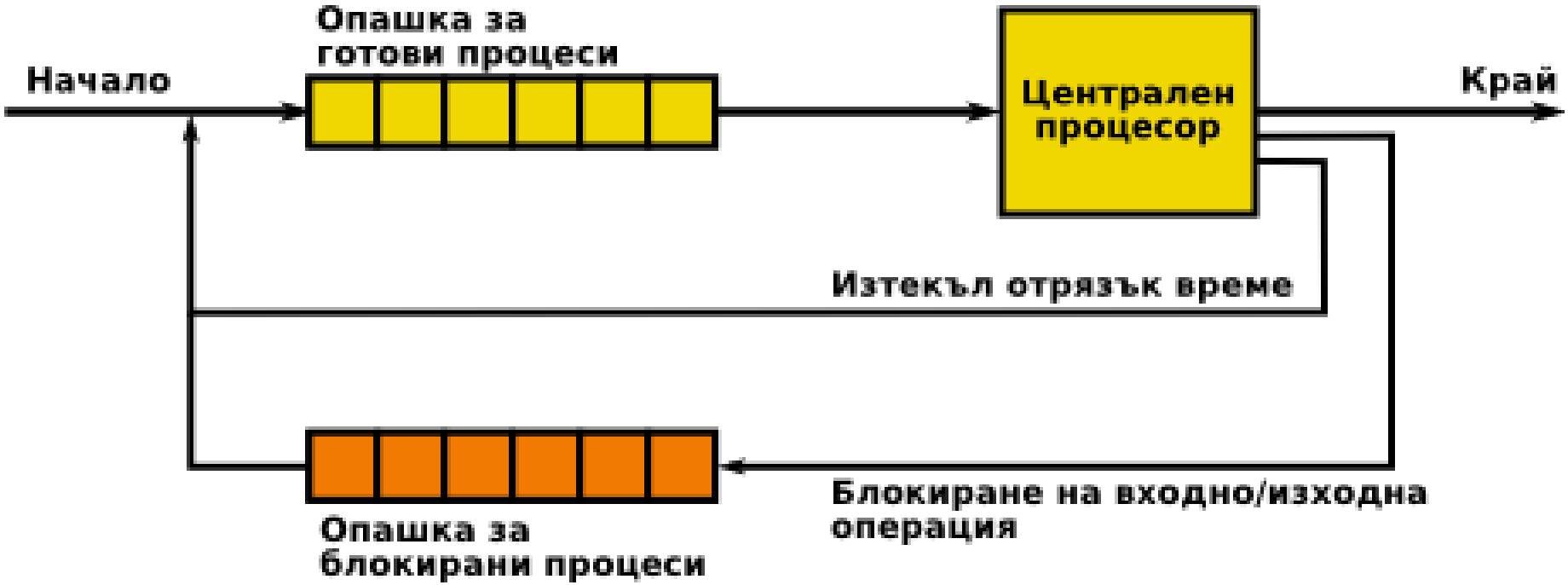
- Този модел е непълен. Може да има различни причини, поради които един процес да бъде “спрян”:
  - Готов за изпълнение.
  - Блокиран върху входно/изходна операция.
- Диспечера не може просто да избере процеса, който е първи в опашката, защото този процес може да е блокиран.

Модел на процеса с пет състояния

- Нов – създаване на процеса.
- Изпълнява се – инструкциите на процеса се изпълняват.
- Блокиран – процесът чака за някакво събитие (прекъсване от входно/изходна операция).
- Готов – процесът е готов за работа и очаква да му бъде отпуснато процесорно време.
- Унищожен – процесът е приключил изпълнението си.



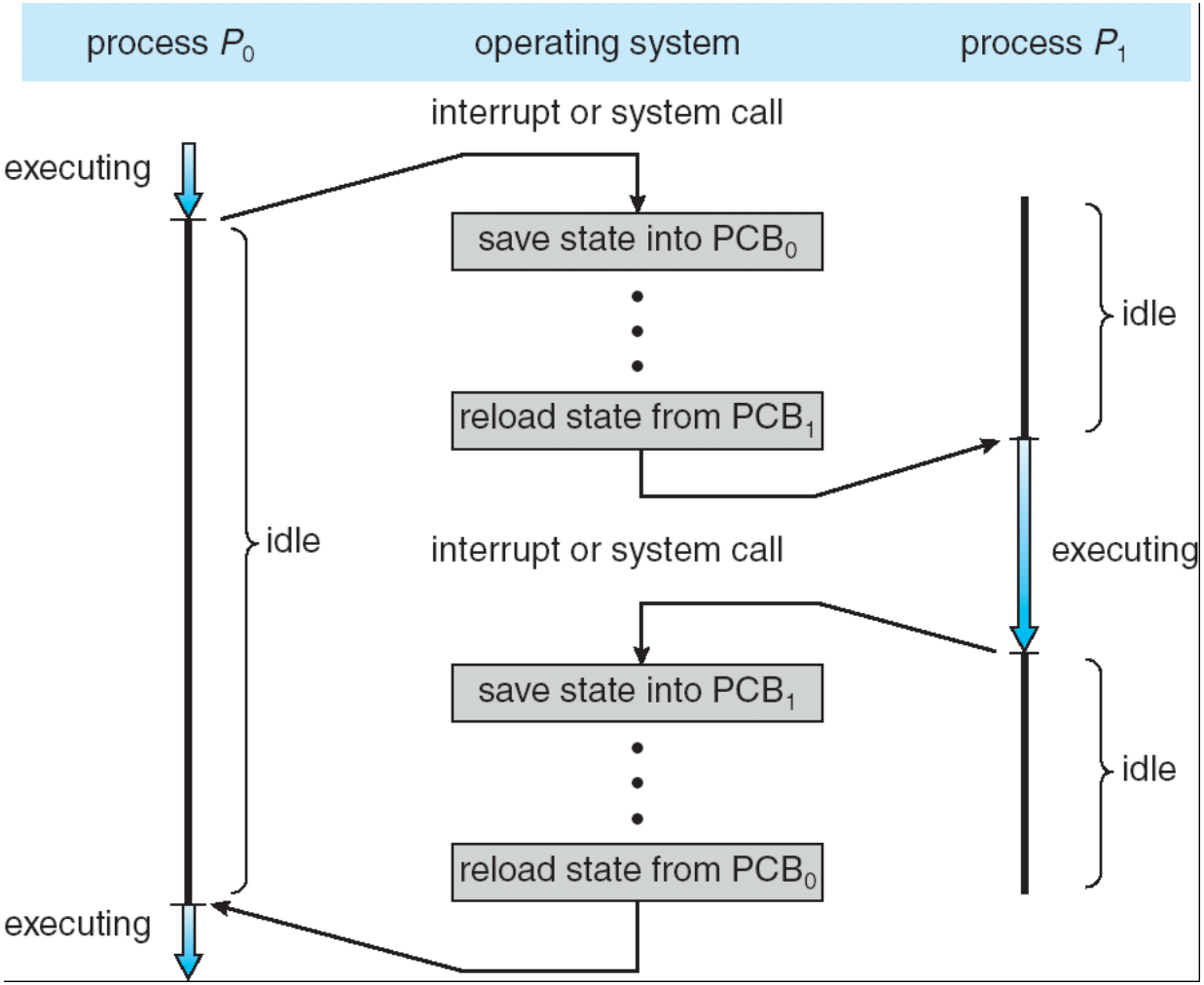
Модел на процеса с пет състояния



Превключване на контекста

- Когато процесорът се превключва от един към друг процес, операционната система трябва да съхрани състоянието на предишния процес и да зареди запазеното състояние на новия процес.
- Превключването на контекста изисква време. Докато се превключва контекста системата не може да изпълнява “полезна” работа.
- Времето за превключване на контекста зависи от хардуерната поддръжка.

Превключване на контекста



Фиг.: Silberschatz, Gavin, Gagne: *Operating Systems Concepts*

Превключване на контекста

Има различни причини, които могат да доведат до превключване на контекста:

- Прекъсване от часовника – процесът е работил максималният допустим отрязък време.
- Входно/изходно прекъсване.
- Страница от паметта е във виртуалната памет и трябва да се качи в оперативната памет.
- Софтуерно прекъсване – грешка при изпълнение на програмата. Може да доведе до унищожаване на процеса.
- Извикване на системна функция.

Превключване на контекста

- Съхраняване на контекста на процесора, включително програмния брояч и другите регистри.
- Обновяване на контролния блок на процеса и промяна на състоянието.
- Преместване на контролния блок на процеса в някоя от опашките от процеси — опашката на готовите процеси, опашката на блокираните процеси и т.н.
- Избиране на следващ процес за изпълнение.
- Обновяване на контролния блок на избрания процес.
- Възстановяване на контекста на избрания процес.

Създаване на процес

- Процесите са организирани в дървовидна структура. Родителският процес създава процеси, които са негови деца. Децата на свой ред могат да създават процеси, които са техни деца.
- При създаването, процесът:
  - получава уникален идентификатор на процеса;
  - заделя се пространство, необходимо за изпълнението на процеса;
  - инициализира се контролния блок на процеса;
  - контролния блок на процеса се добавя във всички необходими таблици, посредством които операционната система контролира работата на процеса.

Създаване на процес: UNIX

- В UNIX за създаване на процес се използва системната функция `fork()`.
- Новосъздадения с `fork()` процес представлява точно копие на родителския процес.
- Трябва да се използва някоя от фамилията системни функции `exec*()` за да се зареди в адресното пространство на процеса нова програма.

Унищожаване на процес

- След като процесът изпълни последния си оператор, той се обръща към операционната система за да я накара да го изтрие. Това се изпълнява със системната функция `exit()`.
- При унищожаването на процеса, операционната система предава информация за това към неговия родителски процес посредством системната функция `wait()`.
- Ресурсите заемани от процеса се освобождават.
- Родителският процес е в състояние да прекрати изпълнението на процесите, които са му деца.
- Когато родителският процес бъде прекратен, операционната система типично унищожава и неговите деца.

Създаване и унищожаване на процес: UNIX

- `pid_t fork (void)`
- `int execv (const char *filename, char *const argv[])`
- `int execl (const char *filename, const char *arg0, . . . )`
- `pid_t waitpid (pid t pid, int *status, int options)`

Създаване на процес: UNIX

```
1 #include <stddef.h>
2 #include <stdlib.h>
3 #include <stdio.h>
4 #include <unistd.h>
5 #include <sys/types.h>
6 #include <sys/wait.h>
7
8 int main () {
9   int status;
10  pid_t pid;
11  pid = fork ();
12  if (pid < 0) {
13    /* The fork failed. Report failure. */
14    status = -1;
15    printf("Unsuccessful␣fork...\n");
16  } else if (pid == 0){
```

Създаване на процес: UNIX

```
16    } else if (pid == 0){
17      /* This is the child process. */
18      //...
19      printf("Hello␣from␣child␣process....\n");
20      exit(0);
21    }
22    else {
23      /* This is the parent process.
24       * Wait for the child to complete. */
25      printf("going␣to␣wait␣for␣child␣process...\n");
26      if (waitpid (pid, &status, 0) != pid) {
27        status = -1;
28      }
29      printf("child␣process␣finished->%d...\n",status);
30    }
31    return status;
32  }
```

Създаване на процес: UNIX

```
1 #include <stddef.h>
2 #include <stdlib.h>
3 #include <stdio.h>
4 #include <unistd.h>
5 #include <sys/types.h>
6 #include <sys/wait.h>
7
8 int main () {
9   int status;
10  pid_t pid;
11  pid = fork ();
12  if (pid < 0) {
13    /* The fork failed. Report failure. */
14    status = -1;
15    printf("Unsuccessful␣fork...\n");
16  } else if (pid == 0){
```

Създаване на процес: UNIX

```
16    } else if (pid == 0){
17      /* This is the child process. */
18      printf("going␣to␣run␣shell␣from␣child...\n");
19      execl("/bin/sh","/bin/sh",NULL);
20    }
21    else {
22      /* This is the parent process.
23       * Wait for the child to complete. */
24      if (waitpid (pid, &status, 0) != pid) {
25        status = -1;
26      }
27      printf("child␣process␣finished␣%d...\n",status);
28    }
29    return status;
30  }
```

Създаване на процес: UNIX

```
lubo@baby ~/school/workspace/os04/code $ ./fork02
going to run shell from child...
sh-3.2$ ls -l
total 24
-rwxr-xr-x 1 lubo wheel 7580 2007-10-23 12:15 fork01
-rw-r--r-- 1 lubo wheel  679 2007-10-23 12:22 fork01.cc
-rwxr-xr-x 1 lubo wheel 7581 2007-10-23 12:44 fork02
-rw-r--r-- 1 lubo wheel  648 2007-10-23 12:44 fork02.cc
sh-3.2$ exit
exit
child process finished 0...
```

Сътрудничество между процеси

- Процесите са изолирани и по принцип те не могат да въздействат на изпълнението на други процеси.
- Сътрудничеството между процеси предполага наличието на механизми, посредством които даден процес да може да въздейства върху изпълнението на друг процес.
- Сътрудничеството между процеси има ред предимства:
  - Увеличаване на производителността.
  - Подобряване на модулността на приложенията.
  - Удобство при разработка.

Производител/потребител

- Основна парадигма за сътрудничество на процеси.
- Процесът *производител* произвежда информацията, която се използва от процеса *потребител*.
- Съществуват две модификации на тази парадигма:
  - Производител/потребител с неограничен буфер – няма граница на размера на буфера.
  - Производител/потребител с ограничен буфер – буфера е с ограничен размер.

Производител/потребител: решение с обща памет

```
1 #define BUFFER_SIZE 10
2 typedef struct {
3   //...
4 } item_t;
5 item_t buffer[BUFFER_SIZE];
6 int in=0;
7 int out=0;
```

Производител

```
1  item_t nextProduced;
2
3  while (1) {
4    while (((in + 1) % BUFFER_SIZE) == out)
5      ; /* do nothing */
6    buffer[in] = nextProduced;
7    in = (in + 1) % BUFFER_SIZE;
8  }
```

Потребител

```
1  item_t nextConsumed;
2
3  while (1) {
4    while (in == out)
5      ; /* do nothing */
6    nextConsumed = buffer[out];
7    out = (out + 1) % BUFFER_SIZE;
8  }
```