Сесия 2 и поправка за “Операционни системи”, СУ, ФМИ, хх.08.2020 г.

Теоретични задачи за КН:

Задача 1. сесия 2 Тройна среща – процесите `P`, `Q` и `R` изпълняват поредица от две инструкции:

```
process P          process Q          process R
  p_1                q_1                r_1
  p_2                q_2                r_2
```

Осигурете чрез семафори синхронизация на `P`, `Q` и `R`, така че:
а) инструкцията `p_1` да се изпълни преди `q_2` и `r_2`
б) инструкцията `q_1` да се изпълни преди `p_2` и `r_2`
в) инструкцията `r_1` да се изпълни преди `p_2` и `q_2`

Задача 1, поправка Всеки от процесите `P`, `Q` и `R` изпълнява поредица от три инструкции:

```
process P          process Q          process R
  p_1                q_1                r_1
  p_2                q_2                r_2
  p_3                q_3                r_3
```

Осигурете чрез семафори синхронизация на `P`, `Q` и `R` така, че да са изпълнени едновременно условията:
(1) инструкция `p_1` да се изпълни преди `q_2` и `r_2`.
(2) инструкция `p_3` да се изпълни след `q_2` и `r_2`.

Задача 1. сесия 1 Множество паралелно работещи копия на всеки от процесите `P`, `Q` и `R` изпълняват поредица от две инструкции:

```
process P          process Q          process R
  p_1                q_1                r_1
  p_2                q_2                r_2
```

Осигурете чрез семафори синхронизация на работещите копия, така че:
а) В произволен момент от времето да работи най-много едно от копията.
б) Работещите копия да се редуват във времето – след изпълнение на копие на `P` да се изпълни копие на `Q`, после копие на `R`. Следва ново изпълнение на `P` и т.н.
в) Първоначално е разрешено да се изпълни копие на `P`.

Задача 2. сесия2 Една от класическите задачи за синхронизация се нарича ”Задача за читателите и писателите”(readers-writers problem).
(а – 5 точки) Опишете условието на задачата.
(б – 10 точки) Опишете решение, използващо семафори.

Задача 2. поправка Опишете реализацията на комуникационна тръба (pipe) чрез семафори. Предполагаме, че тръбата може да съхранява до n байта, подредени в обикновена опашка.

Тръбата се ползва от няколко паралелно работещи изпращачи/получатели на байтове. Процесите изпращачи слагат байтове в края на опашката, получателите четат байтове от началото на опашката.

Теоретични задачи за СИ:

Задача 1. сесия 2 Множество паралелно работещи копия на всеки от процесите `P`, `Q`, `R` и `W` изпълняват поредица от две инструкции:

```
process P          process Q          process R          process W
  p_1                q_1                r_1                w_1
  p_2                q_2                r_2                w_2
```

Осигурете чрез семафори синхронизация на работещите копия, така че:

а) В произволен момент от времето да работи най-много едно от копията.

б) Работещите копия да се редуват във времето – първо се изпълнява копие на `P` или `Q`. След това трябва да се изпълни копие на `R` или `W`. Следва ново изпълнение на копие на `P` или `Q` и т.н.

Задача 1 поправка Всеки от процесите `P`, `Q` и `R` изпълнява поредица от три инструкции:

```
process P                process Q                process R
  p_1                      q_1                      r_1
  p_2                      q_2                      r_2
  p_3                      q_3                      r_3
```

Осигурете чрез семафори синхронизация на `P`, `Q` и `R` така, че да се изпълнят заедно следните изисквания:

(а) Инструкция `p_1` да се изпълни преди `q_2` и `r_2`.

(б) Ако `q_2` се изпълни преди `r_2`, то и `q_3` да се изпълни преди `r_2`.

(в) Ако `r_2` се изпълни преди `q_2`, то и `r_3` да се изпълни преди `q_2`.

Задача 1. сесия 1 Множество паралелно работещи копия на всеки от процесите `P` и `Q` изпълняват поредица от две инструкции:

```
process P                process Q
  p_1                      q_1
  p_2                      q_2
```

Осигурете чрез семафори синхронизация на `P` и `Q`, така че поне една инструкция `p_1` да се изпълни преди всички `q_2`, и поне една инструкция `q_1` да се изпълни преди всички `p_2`.

Задача 2. сесия2 Опишете как се изгражда комуникационен канал (connection) между процес-сървер и процес-клиент със следните системни извиквания в стандарта POSIX:

socket(), bind(), connect(), listen(), accept()

Задача 2. поправка Опишете разликата между синхронни и асинхронни входно-изходни операции.

Дайте примери за програми, при които се налага използването на асинхронен вход-изход.

Примерни решения

Задача 1, поправка, КН За синхронизация използваме семафори `s`, `t` и `u`, инициализираме ги така:

```
semaphore s, t, u
s.init(0)
t.init(0)
u.init(0)
```

Добавяме в кода на процесите `P`, `Q` и `R` синхронизиращи инструкции:

```
process P                process Q                process R
  p_1                      q_1                      r_1
  s.signal()               s.wait()                 s.wait()
  p_2                      s.signal()               s.signal()
  t.wait()                 q_2                      r_2
  u.wait()                 t.signal()               u.signal()
  p_3                      q_3                      r_3
```

Всяка от инструкциите `q_2` и `r_2` може да се изпълни след като съответният процес премине бариерата `s.wait()`.

Това се случва за пръв път след изпълнението на ред `s.signal()` в процеса `P`, който следва инструкция `p_1`. Така изпълнението на `p_1` преди `q_2` и `r_2` е гарантирано.

Да допуснем, че процесът `Q` преминава през инструкцията си `s.wait()` преди процеса `R`. Веднага след това той изпълнява `s.signal()`, което ще позвали и на `R` да премине през своята инструкция `s.wait()`. Така ще се осигури изпълнението и на двете инструкции `q_2` и `r_2`.

Аналогична е ситуацията, когато `R` преминава през `s.wait()` преди процеса `Q`.

Работата със семафорите `t` и `u` осигурява изпълнението на условие (2).

Задача 1, поправка, СИ За синхронизация използваме семафор `t`, инициализираме го с блокиращо начално състояние:

```
semaphore t
t.init(0)
```

Добавяме в кода на процесите `P`, `Q` и `R` синхронизиращи инструкции:

```
process P
  p_1
  t.signal()
  p_2
  p_3
```

```
process Q
  q_1
  t.wait()
  q_2
  q_3
  t.signal()
```

```
process R
  r_1
  t.wait()
  r_2
  r_3
  t.signal()
```

Всяка от инструкциите `q_2` и `r_2` може да се изпълни след като броячът на семафора `t` стане положителен.

Това се случва за пръв път след изпълнението на ред `t.signal()` в процеса `P`, който следва инструкция `p_1`. Така гарантираме изпълнението на изискване (а).

След като броячът на семафора стане 1, един от процесите `Q` и `R` ще достигне пръв до ред `t.wait()` и ще го нулира отново.

Ако процесът `Q` пръв достигне инструкцията `t.wait()`, той ще изпълни редове `q_2` и `q_3`, а процесът `R` ще чака ново увеличение на брояча на семафора, което се случва след изпълнението на последния ред `t.signal()` в процеса `Q`. Така гарантираме изпълнението на изискване (б).

Ако процесът `R` пръв достигне инструкцията `t.wait()`, той ще изпълни редове `r_2` и `r_3`, а процесът `Q` ще чака ново увеличение на брояча на семафора, което се случва след изпълнението на последния ред `t.signal()` в процеса `R`. Така гарантираме изпълнението на изискване (в).

Задача 2. СИ поправка При синхронна входно-изходна операция системното извикване може да доведе до приспиване (блокиране) на потребителския процес, поръчал операцията.

Същевременно, при нормално завършване, потребителският процес разчита на коректно комлектоване на операцията – четене/запис на всички предоставени/поръчани данни във/от входно-изходния канал, или цялостно изпълнение на друг вид операция (примерно, изграждане на TCP връзка).

При асинхронна входно-изходна операция системното извикване не приспива (не блокира) потребителския процес, поръчал операцията.

Същевременно, при невъзможност да се комплектова операцията, ядрото връща управлението на процеса със специфичен код на грешка и друга информация, която служи за определяне на степента на завършеност на операцията.

Потребителският процес трябва да анализира ситуацията и при нужда да направи ново системно повикване по-късно, с цел да довърши операцията.

Използването на асинхронни операции позволява на един процес да извършва паралелна комуникация по няколко канала с различни устройства или процеси, без да бъде блокиран в случай на липса на входни данни, препълване на буфер за изходни данни или друга ситуация, водеща до блокиране.

Типични примери:

(1) Когато ползваме WEB-browser, той трябва да реагира на входни данни от клавиатура и мишка, както и на данните, постъпващи от интернет, т.е. на поне 3 входни канала. Браусерът проверява чрез асинхронни опити за четене по кой от каналите постъпва информация и реагира адекватно.

(2) Сървер в интернет може да обслужва много на брой клиентски програми, като поддържа отворени TCP връзки към всяка от тях. За да обслужва паралелно клиентите, сървърът трябва да ползва асинхронни операции, за да следи по кои връзки протича информация и кои са пасивни.

Когато програмата ползва асинхронни операции и никой от входно-изходните канали не е готов за обмен на данни, тя има нужда от специален механизъм за предоствяне на изчислителния ресурс на останалите процеси. Обикновено в такива случаи програмата се приспива сама за кратък период от време (в UNIX това става с извикване на sleep(), usleep() или nanosleep()).