**Operációs rendszerek BSc**

3. Gyak.

2022. 04. 22.

**Készítette:**

Kopecskó Zsolt Bsc

Mérnökinformatikus

IVO46O

**Miskolc, 2022**

1. Adott négy process a rendszerben, melynek beérkezési sorrendje: A,B,c és D. Minden process USER módban fut és mindegyik processz futásra kész.  
   Kezdetben mindegyik process p\_uspri = 60.  
   Az A,B,C process p\_nice = 0, a D process p\_nice = 5.  
   Mindegyik processz p\_cpu = 0, az óraütés indul, a befejezés legyen 201. óraütésig.

(process\_ütemezés.xlsx)

Round Robin nélkül:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **A process** | | **B process** | | **C process** | | **D process** | | **Az óraütés…** | |
| **Óraütés** | **p\_uspri** | **p\_cpu** | **p\_uspri** | **p\_cpu** | **p\_uspri** | **p\_cpu** | **p\_uspri** | **p\_cpu** | **alatt fut** | **utána fut** |
| **Kiindulás** | **60** | **0** | **60** | **0** | **60** | **0** | **60** | **0** |  | **A** |
| 1 | 60 | 1 | 60 | 0 | 60 | 0 | 60 | 0 | A | A |
| 2 | 60 | 2 | 60 | 0 | 60 | 0 | 60 | 0 | A | A |
| 3 | 60 | 3 | 60 | 0 | 60 | 0 | 60 | 0 | A | A |
| …. | …. | …. | …. | …. | …. | …. | …. | …. | …. | …. |
| 97 | 60 | 97 | 60 | 0 | 60 | 0 | 60 | 0 | A | A |
| 98 | 60 | 98 | 60 | 0 | 60 | 0 | 60 | 0 | A | A |
| 99 | 60 | 99 | 60 | 0 | 60 | 0 | 60 | 0 | A | A |
| 100 | 72 | 86 | 50 | 0 | 50 | 0 | 60 | 0 | A | B |
| 101 | 72 | 86 | 50 | 1 | 50 | 0 | 60 | 0 | B | B |
| 102 | 72 | 86 | 50 | 2 | 50 | 0 | 60 | 0 | B | B |
| …. | …. | …. | …. | …. | …. | …. | …. | …. | …. | …. |
| 196 | 72 | 86 | 50 | 96 | 50 | 0 | 60 | 0 | B | B |
| 197 | 72 | 86 | 50 | 97 | 50 | 0 | 60 | 0 | B | B |
| 198 | 72 | 86 | 50 | 98 | 50 | 0 | 60 | 0 | B | B |
| 199 | 72 | 86 | 50 | 99 | 50 | 0 | 60 | 0 | B | B |
| 200 | 69 | 75 | 72 | 86 | 50 | 0 | 60 | 0 | B | C |
| 201 | 69 | 75 | 72 | 86 | 50 | 1 | 60 | 0 | C | C |

* Az A process kezdi a futást. A futó folyamat minden óraütésnél p\_cpu értéke eggyel nő.
* A 10. óraütésnél mivel nincs magasabb prioritású folyam, így az A process fut tovább.
* A 20. óraütésnél mivel nincs magasabb prioritású folyam, így az A process fut tovább.
* A 30. óraütésnél mivel nincs magasabb prioritású folyam, így az A process fut tovább.
* A 40. óraütésnél mivel nincs magasabb prioritású folyam, így az A process fut tovább.
* A 50. óraütésnél mivel nincs magasabb prioritású folyam, így az A process fut tovább.
* A 60. óraütésnél mivel nincs magasabb prioritású folyam, így az A process fut tovább.
* A 70. óraütésnél mivel nincs magasabb prioritású folyam, így az A process fut tovább.
* A 80. óraütésnél mivel nincs magasabb prioritású folyam, így az A process fut tovább.
* A 90. óraütésnél mivel nincs magasabb prioritású folyam, így az A process fut tovább.
* A 100. óraütésnél a processek p\_uspri és p\_cpu értékeinek karbantartását kell végrehajtani. Mivel 3 futásra kész process van, így a korrekciós faktor értéke KF=(2\*FK)/(2\*FK+1)=(2\*3)/(2\*3+1)=6/7=0,8571
* p\_cpu új értéke: p\_cpu\*KF; p\_uspri=p\_user+p\_cpu/4+2\*p\_nice (p\_user=50); A B process fog a továbbiakban futni, mert annak a legmagasabb a prioritása.
* A 110. óraütésnél mivel nincs magasabb prioritású folyam, így az B process fut tovább.
* A 120. óraütésnél mivel nincs magasabb prioritású folyam, így az B process fut tovább.
* A 130. óraütésnél mivel nincs magasabb prioritású folyam, így az B process fut tovább.
* A 140. óraütésnél mivel nincs magasabb prioritású folyam, így az B process fut tovább.
* A 150. óraütésnél mivel nincs magasabb prioritású folyam, így az B process fut tovább.
* A 160. óraütésnél mivel nincs magasabb prioritású folyam, így az B process fut tovább.
* A 170. óraütésnél mivel nincs magasabb prioritású folyam, így az B process fut tovább.
* A 180. óraütésnél mivel nincs magasabb prioritású folyam, így az B process fut tovább.
* A 190. óraütésnél mivel nincs magasabb prioritású folyam, így az B process fut tovább.
* A 200. óraütésnél a processek p\_uspri és p\_cpu értékeinek karbantartását kell végrehajtani. Mivel 3 futásra kész process van, így a korrekciós faktor értéke KF=(2\*FK)/(2\*FK+1)=(2\*3)/(2\*3+1)=6/7=0,8571
* p\_cpu új értéke: p\_cpu\*KF; p\_uspri=p\_user+p\_cpu/4+2\*p\_nice; A C process fog a továbbiakban futni, mert annak a legmagasabb a prioritása.
* A futási sorrend: [A, B, C, …]

Round Robinnal:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **A process** | | **B process** | | **C process** | | **D process** | | **Az óraütés…** | |  |
| **Óraütés** | **p\_uspri** | **p\_cpu** | **p\_uspri** | **p\_cpu** | **p\_uspri** | **p\_cpu** | **p\_uspri** | **p\_cpu** | **alatt fut** | **utána fut** | **Óraütés** |
| **Kiindulás** | **60** | **0** | **60** | **0** | **60** | **0** | **60** | **0** |  | **A** | **Kiindulás** |
| 1 | 60 | 1 | 60 | 0 | 60 | 0 | 60 | 0 | A | A | 1 |
| 2 | 60 | 2 | 60 | 0 | 60 | 0 | 60 | 0 | A | A | 2 |
| 3 | 60 | 3 | 60 | 0 | 60 | 0 | 60 | 0 | A | A | 3 |
| 4 | 60 | 4 | 60 | 0 | 60 | 0 | 60 | 0 | A | A | 4 |
| 5 | 60 | 5 | 60 | 0 | 60 | 0 | 60 | 0 | A | A | 5 |
| 6 | 60 | 6 | 60 | 0 | 60 | 0 | 60 | 0 | A | A | 6 |
| 7 | 60 | 7 | 60 | 0 | 60 | 0 | 60 | 0 | A | A | 7 |
| 8 | 60 | 8 | 60 | 0 | 60 | 0 | 60 | 0 | A | A | 8 |
| 9 | 60 | 9 | 60 | 0 | 60 | 0 | 60 | 0 | A | A | 9 |
| 10 | 60 | 10 | 60 | 0 | 60 | 0 | 60 | 0 | A | B | 10 |
| 11 | 60 | 10 | 60 | 1 | 60 | 0 | 60 | 0 | B | B | 11 |
| 12 | 60 | 10 | 60 | 2 | 60 | 0 | 60 | 0 | B | B | 12 |
| 13 | 60 | 10 | 60 | 3 | 60 | 0 | 60 | 0 | B | B | 13 |
| 14 | 60 | 10 | 60 | 4 | 60 | 0 | 60 | 0 | B | B | 14 |
| 15 | 60 | 10 | 60 | 5 | 60 | 0 | 60 | 0 | B | B | 15 |
| 16 | 60 | 10 | 60 | 6 | 60 | 0 | 60 | 0 | B | B | 16 |
| 17 | 60 | 10 | 60 | 7 | 60 | 0 | 60 | 0 | B | B | 17 |
| 18 | 60 | 10 | 60 | 8 | 60 | 0 | 60 | 0 | B | B | 18 |
| 19 | 60 | 10 | 60 | 9 | 60 | 0 | 60 | 0 | B | B | 19 |
| 20 | 60 | 10 | 60 | 10 | 60 | 0 | 60 | 0 | B | C | 20 |
| 21 | 60 | 10 | 60 | 10 | 60 | 1 | 60 | 0 | C | C | 21 |
| 22 | 60 | 10 | 60 | 10 | 60 | 2 | 60 | 0 | C | C | 22 |
| 23 | 60 | 10 | 60 | 10 | 60 | 3 | 60 | 0 | C | C | 23 |
| 24 | 60 | 10 | 60 | 10 | 60 | 4 | 60 | 0 | C | C | 24 |
| 25 | 60 | 10 | 60 | 10 | 60 | 5 | 60 | 0 | C | C | 25 |
| 26 | 60 | 10 | 60 | 10 | 60 | 6 | 60 | 0 | C | C | 26 |
| 27 | 60 | 10 | 60 | 10 | 60 | 7 | 60 | 0 | C | C | 27 |
| 28 | 60 | 10 | 60 | 10 | 60 | 8 | 60 | 0 | C | C | 28 |
| 29 | 60 | 10 | 60 | 10 | 60 | 9 | 60 | 0 | C | C | 29 |
| 30 | 60 | 10 | 60 | 10 | 60 | 10 | 60 | 0 | C | D | 30 |
|  | **A process** | | **B process** | | **C process** | | **D process** | | **Az óraütés…** | |  |
| **Óraütés** | **p\_uspri** | **p\_cpu** | **p\_uspri** | **p\_cpu** | **p\_uspri** | **p\_cpu** | **p\_uspri** | **p\_cpu** | **alatt fut** | **utána fut** | **Óraütés** |
| 31 | 60 | 10 | 60 | 10 | 60 | 10 | 60 | 1 | D | D | 31 |
| 32 | 60 | 10 | 60 | 10 | 60 | 10 | 60 | 2 | D | D | 32 |
| 33 | 60 | 10 | 60 | 10 | 60 | 10 | 60 | 3 | D | D | 33 |
| 34 | 60 | 10 | 60 | 10 | 60 | 10 | 60 | 4 | D | D | 34 |
| 35 | 60 | 10 | 60 | 10 | 60 | 10 | 60 | 5 | D | D | 35 |
| 36 | 60 | 10 | 60 | 10 | 60 | 10 | 60 | 6 | D | D | 36 |
| 37 | 60 | 10 | 60 | 10 | 60 | 10 | 60 | 7 | D | D | 37 |
| 38 | 60 | 10 | 60 | 10 | 60 | 10 | 60 | 8 | D | D | 38 |
| 39 | 60 | 10 | 60 | 10 | 60 | 10 | 60 | 9 | D | D | 39 |
| 40 | 60 | 10 | 60 | 10 | 60 | 10 | 60 | 10 | D | A | 40 |
| 41 | 60 | 11 | 60 | 10 | 60 | 10 | 60 | 10 | A | A | 41 |
| 42 | 60 | 12 | 60 | 10 | 60 | 10 | 60 | 10 | A | A | 42 |
| 43 | 60 | 13 | 60 | 10 | 60 | 10 | 60 | 10 | A | A | 43 |
| 44 | 60 | 14 | 60 | 10 | 60 | 10 | 60 | 10 | A | A | 44 |
| 45 | 60 | 15 | 60 | 10 | 60 | 10 | 60 | 10 | A | A | 45 |
| 46 | 60 | 16 | 60 | 10 | 60 | 10 | 60 | 10 | A | A | 46 |
| 47 | 60 | 17 | 60 | 10 | 60 | 10 | 60 | 10 | A | A | 47 |
| 48 | 60 | 18 | 60 | 10 | 60 | 10 | 60 | 10 | A | A | 48 |
| 49 | 60 | 19 | 60 | 10 | 60 | 10 | 60 | 10 | A | A | 49 |
| 50 | 60 | 20 | 60 | 10 | 60 | 10 | 60 | 10 | A | B | 50 |
| 51 | 60 | 20 | 60 | 11 | 60 | 10 | 60 | 10 | B | B | 51 |
| 52 | 60 | 20 | 60 | 12 | 60 | 10 | 60 | 10 | B | B | 52 |
| 53 | 60 | 20 | 60 | 13 | 60 | 10 | 60 | 10 | B | B | 53 |
| 54 | 60 | 20 | 60 | 14 | 60 | 10 | 60 | 10 | B | B | 54 |
| 55 | 60 | 20 | 60 | 15 | 60 | 10 | 60 | 10 | B | B | 55 |
| 56 | 60 | 20 | 60 | 16 | 60 | 10 | 60 | 10 | B | B | 56 |
| 57 | 60 | 20 | 60 | 17 | 60 | 10 | 60 | 10 | B | B | 57 |
| 58 | 60 | 20 | 60 | 18 | 60 | 10 | 60 | 10 | B | B | 58 |
| 59 | 60 | 20 | 60 | 19 | 60 | 10 | 60 | 10 | B | B | 59 |
| 60 | 60 | 20 | 60 | 20 | 60 | 10 | 60 | 10 | B | C | 60 |
| 61 | 60 | 20 | 60 | 20 | 60 | 11 | 60 | 10 | C | C | 61 |
| 62 | 60 | 20 | 60 | 20 | 60 | 12 | 60 | 10 | C | C | 62 |
| 63 | 60 | 20 | 60 | 20 | 60 | 13 | 60 | 10 | C | C | 63 |
| 64 | 60 | 20 | 60 | 20 | 60 | 14 | 60 | 10 | C | C | 64 |
| 65 | 60 | 20 | 60 | 20 | 60 | 15 | 60 | 10 | C | C | 65 |
| 66 | 60 | 20 | 60 | 20 | 60 | 16 | 60 | 10 | C | C | 66 |
| 67 | 60 | 20 | 60 | 20 | 60 | 17 | 60 | 10 | C | C | 67 |
| 68 | 60 | 20 | 60 | 20 | 60 | 18 | 60 | 10 | C | C | 68 |
| 69 | 60 | 20 | 60 | 20 | 60 | 19 | 60 | 10 | C | C | 69 |
| 70 | 60 | 20 | 60 | 20 | 60 | 20 | 60 | 10 | C | D | 70 |
| 71 | 60 | 20 | 60 | 20 | 60 | 20 | 60 | 11 | D | D | 71 |
| 72 | 60 | 20 | 60 | 20 | 60 | 20 | 60 | 12 | D | D | 72 |
| 73 | 60 | 20 | 60 | 20 | 60 | 20 | 60 | 13 | D | D | 73 |
| 74 | 60 | 20 | 60 | 20 | 60 | 20 | 60 | 14 | D | D | 74 |
| 75 | 60 | 20 | 60 | 20 | 60 | 20 | 60 | 15 | D | D | 75 |
| 76 | 60 | 20 | 60 | 20 | 60 | 20 | 60 | 16 | D | D | 76 |
|  | **A process** | | **B process** | | **C process** | | **D process** | | **Az óraütés…** | |  |
| **Óraütés** | **p\_uspri** | **p\_cpu** | **p\_uspri** | **p\_cpu** | **p\_uspri** | **p\_cpu** | **p\_uspri** | **p\_cpu** | **alatt fut** | **utána fut** | **Óraütés** |
| 77 | 60 | 20 | 60 | 20 | 60 | 20 | 60 | 17 | D | D | 77 |
| 78 | 60 | 20 | 60 | 20 | 60 | 20 | 60 | 18 | D | D | 78 |
| 79 | 60 | 20 | 60 | 20 | 60 | 20 | 60 | 19 | D | D | 79 |
| 80 | 60 | 20 | 60 | 20 | 60 | 20 | 60 | 20 | D | A | 80 |
| 81 | 60 | 21 | 60 | 20 | 60 | 20 | 60 | 20 | A | A | 81 |
| 82 | 60 | 22 | 60 | 20 | 60 | 20 | 60 | 20 | A | A | 82 |
| 83 | 60 | 23 | 60 | 20 | 60 | 20 | 60 | 20 | A | A | 83 |
| 84 | 60 | 24 | 60 | 20 | 60 | 20 | 60 | 20 | A | A | 84 |
| 85 | 60 | 25 | 60 | 20 | 60 | 20 | 60 | 20 | A | A | 85 |
| 86 | 60 | 26 | 60 | 20 | 60 | 20 | 60 | 20 | A | A | 86 |
| 87 | 60 | 27 | 60 | 20 | 60 | 20 | 60 | 20 | A | A | 87 |
| 88 | 60 | 28 | 60 | 20 | 60 | 20 | 60 | 20 | A | A | 88 |
| 89 | 60 | 29 | 60 | 20 | 60 | 20 | 60 | 20 | A | A | 89 |
| 90 | 60 | 30 | 60 | 20 | 60 | 20 | 60 | 20 | A | B | 90 |
| 91 | 60 | 30 | 60 | 21 | 60 | 20 | 60 | 20 | B | B | 91 |
| 92 | 60 | 30 | 60 | 22 | 60 | 20 | 60 | 20 | B | B | 92 |
| 93 | 60 | 30 | 60 | 23 | 60 | 20 | 60 | 20 | B | B | 93 |
| 94 | 60 | 30 | 60 | 24 | 60 | 20 | 60 | 20 | B | B | 94 |
| 95 | 60 | 30 | 60 | 25 | 60 | 20 | 60 | 20 | B | B | 95 |
| 96 | 60 | 30 | 60 | 26 | 60 | 20 | 60 | 20 | B | B | 96 |
| 97 | 60 | 30 | 60 | 27 | 60 | 20 | 60 | 20 | B | B | 97 |
| 98 | 60 | 30 | 60 | 28 | 60 | 20 | 60 | 20 | B | B | 98 |
| 99 | 60 | 30 | 60 | 29 | 60 | 20 | 60 | 20 | B | B | 99 |
| 100 | 57 | 26 | 57 | 26 | 54 | 17 | 64 | 17 | B | C | 100 |
| 101 | 57 | 26 | 57 | 26 | 54 | 18 | 64 | 17 | C | C | 101 |
| 102 | 57 | 26 | 57 | 26 | 54 | 19 | 64 | 17 | C | C | 102 |
| 103 | 57 | 26 | 57 | 26 | 54 | 20 | 64 | 17 | C | C | 103 |
| … | … | … | … | … | … | … | … | … | … | ... |  |
| 197 | 57 | 26 | 57 | 26 | 54 | 114 | 64 | 17 | C | C | 197 |
| 198 | 57 | 26 | 57 | 26 | 54 | 115 | 64 | 17 | C | C | 198 |
| 199 | 57 | 26 | 57 | 26 | 54 | 116 | 64 | 17 | C | C | 199 |
| 200 | 56 | 22 | 56 | 22 | 75 | 100 | 64 | 15 | C | A | 200 |
| 201 | 56 | 23 | 56 | 22 | 75 | 100 | 64 | 15 | A | A | 201 |

* Az A process kezdi a futást. A futó folyamat minden óraütésnél p\_cpu értéke eggyel nő.
* A 10. óraütésnél mivel van azonos prioritású futásra kész process ezért váltás történik, így a B process fut tovább.
* A 20. óraütésnél mivel van azonos prioritású futásra kész process ezért váltás történik, így a C process fut tovább.
* A 30. óraütésnél mivel van azonos prioritású futásra kész process ezért váltás történik, így a D process fut tovább.
* A 40. óraütésnél mivel van azonos prioritású futásra kész process ezért váltás történik, így a A process fut tovább.
* A 50. óraütésnél mivel van azonos prioritású futásra kész process ezért váltás történik, így a B process fut tovább.
* A 60. óraütésnél mivel van azonos prioritású futásra kész process ezért váltás történik, így a C process fut tovább.
* A 70. óraütésnél mivel van azonos prioritású futásra kész process ezért váltás történik, így a D process fut tovább.
* A 80. óraütésnél mivel van azonos prioritású futásra kész process ezért váltás történik, így a A process fut tovább.
* A 90. óraütésnél mivel van azonos prioritású futásra kész process ezért váltás történik, így a B process fut tovább.
* A 100. óraütésnél a processek p\_uspri és p\_cpu értékeinek karbantartását kell végrehajtani. Mivel 3 futásra kész process van, így a korrekciós faktor értéke KF=(2\*FK)/(2\*FK+1)=(2\*3)/(2\*3+1)=6/7=0,8571
* p\_cpu új értéke: p\_cpu\*KF; p\_uspri=p\_user+p\_cpu/4+2\*p\_nice (p\_user=50); A C process fog a továbbiakban futni.
* A 110. óraütésnél nincs az éppen futó procesel azonos prioritású futásra kész process, így a 200. óraütésnél esedékes p\_uspri újraszámításig csak a C process fog futni.
* A 200. óraütésnél a processek p\_uspri és p\_cpu értékeinek karbantartását kell végrehajtani. Mivel 3 futásra kész process van, így a korrekciós faktor értéke KF=(2\*FK)/(2\*FK+1)=(2\*3)/(2\*3+1)=6/7=0,8571
* p\_cpu új értéke: p\_cpu\*KF; p\_uspri=p\_user+p\_cpu/4+2\*p\_nice; Az A process fog a továbbiakban futni.
* A futási sorrend: [A, B, C, D, A, B, C, D, A, B, C, A, …]

1. Készítse el a következő feladatot, melyben egy szignálkezelő több szignált is tud kezelni:
2. Készítsen egy szignálkezelőt (handleSignals), amely a SIGINT (CTRL +C) vagy SIGQUIT (CTRL +\) jelek fogására vagy kezelésére képes.
3. Ha a felhasználó SIGQUIT jelet generál, akkor a kezelő egyszerűen kiírja az üzenetet visszatérési értékét.
4. Ha a felhasználó először generálja a SIGINT jelet, akkor a jelet úgy módosítja, hogy a következő alkalommal alapértelmezett műveletet hajtson végre (SIG\_DFL).
5. Ha a felhasználó másodszor generálja a SIGINT jelet, akkor végrehajt egy alapértelmezett műveletet, amely a program befejezése.

(IVO46O\_tobbsignal.c)

#include <stdlib.h>

#include <stdio.h>

#include <signal.h>

#include <unistd.h>

/\* A signal kezelő deklarálása \*/

void handleSignals(int sig\_num){

if(sig\_num == 2){

signal(SIGINT,SIG\_DFL); // a SIGINT vezérlését alaphelyzetbe állítja

printf("A következő SIGINT jelre a program leáll! [%d]\n",sig\_num);

}else

printf("Ez nem SIGINT jel! [%d]\n",sig\_num);

fflush(stdout); //kiüríti az outpt buffert

}

int main(){

signal(SIGINT, handleSignals); //SIGINT jelre átadja a vezérlést a signálkezelőnek, a jel értékével

signal(SIGQUIT, handleSignals); //SIGQUIT jelre átadja a vezérlést a signálkezelőnek, a jel értékével

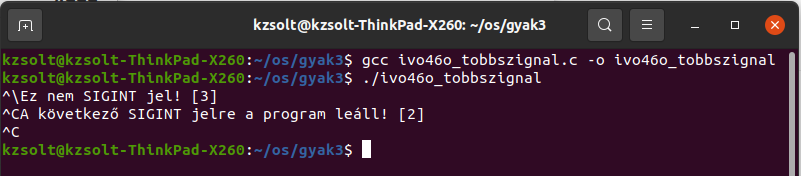
/\* végtelen ciklus a semmittevéshez \*/

for ( ;; )

pause();

return 0;

}



1. Készítsen C programot, ahol egy szülő process létrehoz egy csővezetéket, a gyerek process beleír egy szöveget a csővezetékbe, a szülő process ezt kiolvassa és kiírja.

(IVO46O\_unnamed.c)

int main(){

char rd\_buffer[30]; //az bemeneti buffer méretének megadása

char msg[] = "Kopecskó Zsolt IVO46O"; // a pipe-ba beírandó szöveg

int pipefds[2], childpid, nbytes;

if (pipe(pipefds) < 0){ //A pipe létrejöttének leellenőrzése!

printf("Nem sikerült a pipe-ot létrehozni!\n");

exit(1);

}

if ((childpid = fork()) == -1){ //A gyermek process létrejöttének leellenőrzése!

printf("Nem sikerült a gyermek process létrehozása!\n");

exit(1);

}

if (childpid == 0){

/\*Gyerek process beír a pipe-ba.\*/

close(pipefds[0]); //A pipe olvásási végének bezárása, az írás előtt!

printf("Gyerek process beír a pipe-ba.\n");

write(pipefds[1], msg, (strlen(msg))); //az üzenet beírása a pipe-ba

exit(0); //sikeres lefutás után 0 értéket ad vissza a szülő processnek.

}

else {

/\*Szülő process megvárja míg a gyermek process befejeződik,

majd kiolvas az adatokat a pipe-ból.\*/

wait(0);

close(pipefds[1]); //a pipe beírási végének lezárása

printf("A szülő process kiolvassa a pibe-ba írt üzenetet!\n");

while ((nbytes = read(pipefds[0], rd\_buffer, sizeof(rd\_buffer))) > 0)

printf("[%s]\n", rd\_buffer);

if (nbytes != 0) // Ha nem ürül ki a pipe akkor 2 érték

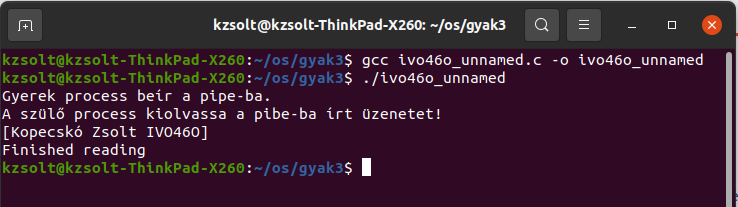
exit(2); // visszaadásával terminálódik a program.

printf("Finished reading\n"); //A sikeres kiolvasás visszaigazolása.

}

return 0;

}



1. Készítsen C programot, ahol a szülő process létrehoz egy nevesített csővezetéket, a gyermek process beleír egy szöveget a csővezetékbe, a szülő ezt kiolvassa és kiírja.

(IVO46O\_named.c)

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <wait.h>

#include <fcntl.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <unistd.h>

int main(){

int fd, pid;

char rd\_buffer[20]; //az olvasási buffer méretének megadása

char msg[20] = "Kopecskó Zsolt"; //a gyermek process által elhelyezendő adat

char \* myfifo = "/home/kzsolt/os/gyak3/ivo46o"; //FIFO file elérési útvonala és neve

int test = mkfifo(myfifo, 0666); // FIFO file létrehozása

if(test == -1){ //vizsgélat, hogy létezik-e a FIFO file

perror("hiba történt a FIFO létrehozása közben - a file lehetséges, hogy már létezik\n");

unlink("ivo46o"); //törli a létező FIFO filet

test = mkfifo(myfifo, 0666); // létrehozza a FIFO filet

printf("A létező FIFO file törölve lett és egy új lett létrehozva!\n\n");

}

if ((pid = fork()) == -1){ //A gyermek process létrejöttének leellenőrzése!

printf("Nem sikerült a gyermek process létrehozása!\n");

exit(1);

}

if (pid == 0){

/\*Gyerek process beír a pipe-ba.\*/

printf("Gyerek process beír a pipe-ba.\nGyerek PID: [%d] ---> Szülő PID: [%d]\n\n",getpid(), getppid());

fd = open(myfifo, O\_WRONLY); //A FIFO file megnyitása írásra.

if(fd == -1){ //vizsgálat, hogy megnyitható-e a FIFO file

perror("Nem lehet megnyitni a fifo file-t!");

exit(1);

}

write(fd, msg, strlen(msg)); //a megadott adat beírása a FIFO file-ba.

close(fd); //FIFO file bezárása

exit(0); //sikeres lefutás után 0 értéket ad vissza a szülő processnek.

}else {

/\*Szülő process megvárja míg a gyermek process befejeződik,

majd kiolvas az adatokat a pipe-ból.\*/

fd = open(myfifo, O\_RDONLY); //FIFO file megnyitása olvasásra.

wait(0); //A gyermek process lefutásának megvárása.

printf("A szülő process kiolvassa a pibe-ba írt üzenetet! Szülő PID: [%d]\n\n", getpid());

read(fd, rd\_buffer, sizeof(rd\_buffer)); //FIFO file tartalmának kiolvasása

printf("[%s]\n\n", rd\_buffer); //A kiolvasott adat kiírása

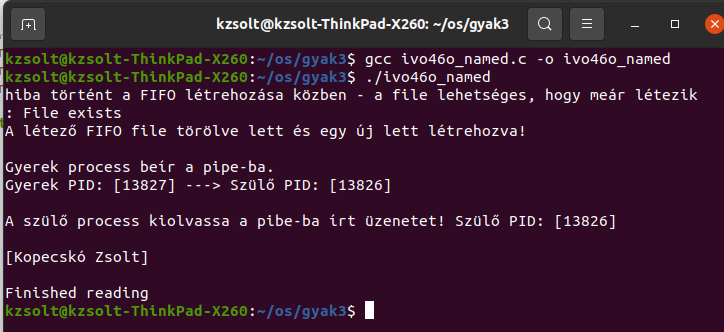
close(fd); // FIFO file bezárása

printf("Finished reading\n"); //A sikeres kiolvasás visszaigazolása.

}

return 0;

}



1. Adott egy rendszerben az összes osztály-erőforrás száma: (R1:10, R2:9, R3:12)

A rendszerben 4 process van (P1, P2, P3, P4)

Biztonságos-e holtpontmentesség szempontjából a rendszer – a következő kiindulási állapot alapján?

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Maximális Igény | | | |  | Foglalási Igény | | | |
|  | R1 | R2 | R3 |  |  | R1 | R2 | R3 |
| 10 | 9 | 12 |  | 10 | 9 | 12 |
| P1 | 4 | 4 | 5 |  | P1 | 2 | 2 | 3 |
| P2 | 1 | 4 | 3 |  | P2 | 1 | 2 | 2 |
| P3 | 6 | 7 | 7 |  | P3 | 0 | 1 | 3 |
| P4 | 3 | 7 | 10 |  | P4 | 2 | 1 | 2 |

1. határozza a folyamtok által igényelt erőforrások mátrixát?

IGÉNY = MAX.IGÉNY – FOGLALÁSI.IGÉNY

|  |  |  |  |
| --- | --- | --- | --- |
|  | R1 | R2 | R3 |
| P1 | 4-2=2 | 4-2=2 | 5-3=2 |
| P2 | 1-1=0 | 4-2=2 | 3-2=1 |
| P3 | 6-0=6 | 7-1=6 | 7-3=4 |
| P4 | 3-2=1 | 7-1=6 | 10-2=8 |

1. Határozza meg a pillanatnyilag szabad erőforrások számát?

Pill.szabad.erőforr=erőforrás kezdőértéke – a folyamtok által lefoglalt értékerk

|  |  |  |  |
| --- | --- | --- | --- |
|  | R1 | R2 | R3 |
| Szabad erőf. | 10-2-1-0-2=5 | 9-2-2-1-1=3 | 12-3-2-3-2=2 |

1. Igazolja az processek végrehajtásának lehetséges sorrendjét:

(Bankár algoritmus)

A bankár algoritmus úgy kerüli el a holtpont kialakulásának lehetőségét, hogy a rendszert mindig ún. biztonságos állapotban tartja. Egy állapotot akkor tekintünk biztonságosnak, ha létezik legalább egy olyan sorozat, amely szerint az összes folyamat erőforrás igénye kielégíthető.

Az algoritmus úgy működik, hogy egy beérkezett erőforrásigény teljesítése előtt az erőforrás kezelő kiszámolja, hogy ha az igényt *teljesítené,* akkor a rendszer biztonságos állapotban maradna-e. Ha igen, teljesíti az igényt; ha nem, akkor a folyamat várakozó listára kerül. Ha egy folyamat visszaad erőforrásokat, akkor az erőforrás kezelő végignézi a várakozó listán levő folyamatokat, hogy most már kielégíthető-e valamelyik igénye.

Az algoritmus biztosítja a holtpontmentességet, ugyanis az erőforrás-kezelő mindaddig nem elégíti ki a várakozó folyamatok igényét, amíg nem biztosítható, hogy az újabb erőforrások lefoglalása által előálló új állapot biztonságos, azaz a futó folyamatok erőforrás igényét valamilyen sorrendben ki tudjuk elégíteni, és ezáltal azok le tudnak futni.

A sorend kiszámításának lépései: (bankár\_algrtms.xlsx)

* A fentebb már kiszámított pillanatnyilag szabad erőforrásokból megvizsgáljuk, hogy melyik process erőforrásigénye teljesíthető. Látható, hogy bármelyik. Kiválasztásra kerül az P1 process. Lefutása után az process foglalási igénye hozzáadódik a szabad erőforrásokhoz.

Ezáltal az új szabad erőforrások:

|  |  |  |  |
| --- | --- | --- | --- |
| P1 után | R1 | R2 | R3 |
| Szabad erőf. | 5+2=7 | 3+2=5 | 2+3=5 |

* Az új szabad erőforrások ismeretében újra megvizsgáljuk a még nem lefutott processeket, hogy melyik teljesíthető. Akármelyik process lefutásához van a rendszerben elegendő erőforrás. Futásra kiválasztjuk a P2 processt. Lefutása után a folyamat foglalási igénye hozzáadódik a szabad erőforrásokhoz.

Ezáltal az új szabad erőforrások:

|  |  |  |  |
| --- | --- | --- | --- |
| P2 után | R1 | R2 | R3 |
| Szabad erőf. | 7+1=8 | 2+5=7 | 2+5=7 |

* Az új szabad erőforrások ismeretében újra megvizsgáljuk a még nem lefutott processeket, hogy melyik teljesíthető. Akármelyik process lefutásához van a rendszerben elegendő erőforrás. Futásra kiválasztjuk a P3 processt. Lefutása után a folyamat foglalási igénye hozzáadódik a szabad erőforrásokhoz.

Ezáltal az új szabad erőforrások:

|  |  |  |  |
| --- | --- | --- | --- |
| P3 után | R1 | R2 | R3 |
| Szabad erőf. | 0+8=8 | 1+7=8 | 3+7=10 |

* Az új szabad erőforrások ismeretében újra megvizsgáljuk a még nem lefutott processeket, hogy melyik teljesíthető. A rendszerben elegendő erőforrás van, hogy az utolsó folyamatot kiszolgálja. Lefutása után a folyamat foglalási igénye hozzáadódik a szabad erőforrásokhoz és ezáltal megkapjuk az erőforrások kiindulási értékeit, mivel nincs folyamat, ami foglalná azokat.

Ezáltal az új szabad erőforrások:

|  |  |  |  |
| --- | --- | --- | --- |
| P4 után | R1 | R2 | R3 |
| Szabad erőf. | 2+8=10 | 1+8=9 | 2+10=12 |

* A fentiek alapján a folyamtok lefutásának egy lehetséges sorrendje:

[P1, P2, P3, P4]