Name: Vipin Nagar

ID: 11700917

Roll No: 47

Email Address: vnagar@iitk.ac.in

GitHub Link: <https://github.com/vnagar007/Operating-System>

**Problem:**

1. There are 3 student processes and 1 teacher process. Students are supposed to do their assignments and they need 3 things for that pen, paper and question paper. The teacher has an infinite supply of all the three things. One students has pen,an other has paper and another has question paper. The teacher places two things on a shared table and the student having the third complementary thing makes the assignment and tells the teacher on completion. The teacher then places another two things out of the three and again the student having the third thing makes the assignment and tells the teacher on completion. This cycle continues. WAP to synchronize the teacher and the students.

**Explanation:**

1. In the given solution I have taken all the student processes and resources in 2D array and initialized then to 0.
2. I have made 3 student processes in three different functions which will be executed by single s\_thread and one t\_thread for execution of teacher process.
3. User will get a menu to select any two out of three resources that are to be placed on shared table.
4. If one process is completed there will be a message printed on the screen saying process is completed.
5. When one process is executing no other student or teacher process will execute and for achieving this I have used Mutex lock.
6. When a process starts to execute it acquires the lock and when it completes the execution releases the lock.
7. After completion of all the three processes the program will end.

**Code snippet:**

|  |  |
| --- | --- |
| 1. | #include<stdio.h> |
| 2. | #include<stdlib.h> |
| 3. | #include<pthread.h> |
| 4. | #include<stdbool.h> |
| 5. |  |
| 6. | int student[3][4]={0}; |
| 7. | void \*teacher(); |
| 8. | void \*student1(); |
| 9. | void \*student2(); |
| 10. | void \*student3(); |
| 11. | pthread\_mutex\_t lck; |
| 12. | int ch1,ch2; |
| 13. | int r1,r2; |
| 14. |  |
| 15. | int main() |
| 16. | { |
| 17. | printf("\t\t\t---Welcome---\n"); |
| 18. | pthread\_mutex\_init(&lck,NULL); |
| 19. |  |
| 20. | student[1][1]=1; |
| 21. | student[2][2]=2;student[3][3]=1; |
| 22. | pthread\_t t\_thread; |
| 23. | pthread\_t s\_thread; |
| 24. | printf("Resources Menu: \n\t\tPress '1' for pen\n\t\tPress '2' for paper \n\t\tPress '3' for question\_paper \n"); |
| 25. |  |
| 26. | while(1) |
| 27. | { |
| 28. | if(student[1][4]==1 && student[2][4]==1 && student[3][4]==1){break;} |
| 29. | pthread\_create(&t\_thread, NULL, teacher, NULL); |
| 30. | pthread\_join(t\_thread,NULL); |
| 31. | if((ch1==1 && ch2==2 || ch2==1 && ch1==2 ) && student[3][4]==0) |
| 32. | { |
| 33. | pthread\_create(&s\_thread, NULL, student3, NULL); |
| 34. | pthread\_join(s\_thread,NULL); |
| 35. | } |
| 36. | else if((ch1==1 && ch2==3 || ch2==1 && ch1==3 ) && student[2][4]==0) |
| 37. | { |
| 38. | pthread\_create(&s\_thread, NULL, student2, NULL); |
| 39. | pthread\_join(s\_thread,NULL); |
| 40. | } |
| 41. | else if((ch1==2 && ch2==3 || ch2==2 && ch1==3 ) && student[1][4]==0) |
| 42. | { |
| 43. | pthread\_create(&s\_thread, NULL, student1, NULL); |
| 44. | pthread\_join(s\_thread,NULL); |
| 45. | } |
| 46. | else |
| 47. | { |
| 48. | printf("\n\tError (007): try again.. with different choices.\n"); |
| 49. | } |
| 50. | } |
| 51. | printf("\n\t----Done---\n"); |
| 52. |  |
| 53. |  |
| 54. | } |
| 55. |  |
| 56. | void \*teacher() |
| 57. | { |
| 58. | pthread\_mutex\_lock(&lck); |
| 59. | printf("\nFirst Resource on shared tabel:-\t"); |
| 60. | scanf("%d",&ch1); |
| 61. | printf("Second Resource on shared tabel:-\t"); |
| 62. | scanf("%d",&ch2); |
| 63. | pthread\_mutex\_unlock(&lck); |
| 64. | } |
| 65. |  |
| 66. | void \*student2() |
| 67. | { |
| 68. | pthread\_mutex\_lock(&lck); |
| 69. | printf("\nChoices Made = 'pen', 'question\_paper'\n"); |
| 70. | student[2][4]=1; |
| 71. | printf("\n\tStudent 2 has Completed the assignment. \n"); |
| 72. | pthread\_mutex\_unlock(&lck); |
| 73. | } |
| 74. | void \*student3() |
| 75. | { |
| 76. | pthread\_mutex\_lock(&lck); |
| 77. | printf("\nChoices Made = 'pen', 'paper'\n"); |
| 78. | student[3][4]=1; |
| 79. | printf("\n\tStudent 3 has Completed the assignment.\n"); |
| 80. | pthread\_mutex\_unlock(&lck); |
| 81. | } |
| 82. | void \*student1() |
| 83. | { |
| 84. | pthread\_mutex\_lock(&lck); |
| 85. | printf("\nChoices Made = 'paper', 'question\_paper'\n"); |
| 86. | student[1][4]=1; |
| 87. | printf("\n\tStudent 1 has Completed the assignment.\n"); |
| 88. | pthread\_mutex\_unlock(&lck); |
| 89. | } |

**Problem:**

1. Two types of people can enter into a library- students and teachers. After entering the library, the visitor searches for the required books and gets them. In order to get them issued, he goes to the single CPU which is there to process the issuing of books. Two types of queues are there at the counter-one for students and one for teachers. A student goes and stands at the tail of the queue for students and similarly the teacher goes and stands at the tail of the queue for teachers (FIFO). If a student is being serviced and a teacher arrives at the counter, he would be the next person to get service (PRIORITY-non preemptive). If two teachers arrive at the same time, they will stand in their queue to get service (FIFO). WAP to ensure that the system works in a non-chaotic manner.

* If a teacher is being served and during the period when he is being served, another teacher comes, then that teacher would get the service next. This process might continue leading to increase in waiting time of students. Ensure in your program that the waiting time of students is minimized.

**Code snippet:**

|  |  |
| --- | --- |
| 1. | #include<stdio.h> |
| 2. | int main() |
| 3. | { |
| 4. | int p[20],bt[20], su[20], wt[20],tat[20],i, k, n, temp; |
| 5. | float wtavg, tatavg; |
| 6. | printf("Enter the number of PROCESS in the queue --- "); |
| 7. | scanf("%d",&n); |
| 8. | for(i=0;i<n;i++) |
| 9. | { |
| 10. | p[i] = i; |
| 11. | printf("Enter the Burst Time for process %d --- ", i); |
| 12. | scanf("%d",&bt[i]); |
| 13. | printf("teacher/student process (0/1) ? --- "); |
| 14. | scanf("%d", &su[i]); |
| 15. | } |
| 16. |  |
| 17. | for(i=0;i<n;i++) |
| 18. | { |
| 19. | for(k=i+1;k<n;k++) |
| 20. | { |
| 21. | if(su[i] > su[k]) |
| 22. | { |
| 23. | temp=p[i]; |
| 24. | p[i]=p[k]; |
| 25. | p[k]=temp; |
| 26. |  |
| 27. | temp=bt[i]; |
| 28. | bt[i]=bt[k]; |
| 29. | bt[k]=temp; |
| 30. |  |
| 31. | temp=su[i]; |
| 32. | su[i]=su[k]; |
| 33. | su[k]=temp; |
| 34. | } |
| 35. | } |
| 36. | } |
| 37. | wtavg = wt[0] = 0; |
| 38. | tatavg = tat[0] = bt[0]; |
| 39. | for(i=1;i<n;i++) |
| 40. | { |
| 41. | wt[i] = wt[i-1] + bt[i-1]; |
| 42. | tat[i] = tat[i-1] + bt[i]; |
| 43. | wtavg = wtavg + wt[i]; |
| 44. | tatavg = tatavg + tat[i]; |
| 45. | } |
| 46. | printf("\nPROCESS\t\t TEACHER/STUDENT PROCESS \tBURST TIME\tWAITING TIME\tTURNAROUND TIME"); |
| 47. |  |
| 48. | for(i=0;i<n;i++) |
| 49. | { |
| 50. | printf("\n%d \t\t %d \t\t %d \t\t %d \t\t\t %d ",p[i],su[i],bt[i],wt[i],tat[i]); |
| 51. | } |
| 52. |  |
| 53. | printf("\nAverage Waiting Time is --- %f",wtavg/n); |
| 54. | printf("\nAverage Turnaround Time is --- %f",tatavg/n); |
| 55. |  |
| 56. | return 0; |
| 57. | } |