National University of Computer and Emerging Sciences, Lahore Campus

|  | Course:  Program:  Duration:  Paper Date:  Section:  Exam: | Operating System  BS(Computer Science)  3 hour  27*th* December, 2017  Final | Course Code:  Semester:  Total Marks:  Weight:  Page(s):  Roll No. | CS-205  Fall 2017  54  45%  3 |
| --- | --- | --- | --- | --- |

Instructions/Notes: Answer questions on the question paper. Write answers clearly and precisely, if the answers are not easily readable then it will result in deduction of marks. Use extra sheet for rough work, cutting and blotting on this sheet will result in deduction of marks.

Question 1 (10 points): Although practically it is impossible to implement shortest job first algorithm, but if we had following class implementations, we could easily implement SJF. So lets do it. Hint: read the declarations carefully!

| class List { *// it is the list of all processes ready to run .*  public :  bool addToList ( int element ); *// adds a process described by element to the list . Return value is not used here .*  bool removeFromList ( int element ); *// removes a process described by element to the list . Returns value is not used here .*  };  class Iterator {  public :  Iterator ( List list );*// initializes the iterator with the list .*  int getNext () ; *// iterates over the process list , just like an iterator in STL . Returns -1 when reaches the end . Othewise returns the PID and moves next .* friend class List ;  };  *// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -* double getProcessRemaningTime ( int pid ) *// returns the time the process , described by the pid , will take in the next burst . Based on the return value of this fucntion we can make decisions .* |
| --- |

int getNextProcessToRun ( int leavingProcessID , List list ) *// First parameter is the process which is leaving the CPU . Second the list of ready processes . The function returns the ID of the process to run next .*

{

Iterator iter = new Iterator ( list );

int pid = -1;

int nextPID = leavingProcessID ;

double remainingTime = DBL\_MAX ;

double thisTime = 0;

while (( pid = Iterator . getNext () ) > -1)

{

if (( thisTime = getProcessRemaningTime ( pid )) < remainingTime )

{

nextPID = pid ;

remainingTime = thisTime ;

}

}

if ( pid > -1)

{

list . removeFromList ( nextPID );

*// list . addToList ( l e a v i n g P r o c e s s I D ); that is error , as SJF is non - preemptive* }

return nextPID ;

}

School of Computer Science Page 1 of 3

Question 2 (10 points): Implement a function which takes the logical address and returns the physical address. Use the functions provided below. Hint: read the declarations carefully!

| int getPageNumber ( int logicalAddress ); *// takes logical address and returns the associated page number .*  int getFrameNumber ( int pageNumber ); *// takes the pagenumber and returns the associated frame number .*  int loadPageInMemory ( int pageNumber ); *// loads a page from backing store into the physical memory , and return the framenumber where the page was loaded .*  void setFrameNumber ( int pagenumber , int framenumber ); *// sets the framenumber of the pagenumber . Also sets all relevant bits of the page table .*  int replacePageByFrameNumber ( int logicaladdress , int framenumber ); *// converts the l o g i c a l a d d r e s s into a physical address by replacing the page number by frame number .* |
| --- |

int getPhysicalAddress ( int logicalAddress )

{

int pn = getPageNumber ( logicalAddress );

int fn= -1;

if ( getFrameNumber (pn) < 0)

{

int fn = loadPageInMemory (pn);

setFrameNumber (pn ,fn);

}

if (fn == -1)

{

*// PANIC*

}

return replacePageByFrameNumber ( logicaladdress ,fn);

}

Question 3 (10 points): Get the physical byte stored in a file which exists in a file system that uses single indexed table. Parameters are the logical address of the byte, and the file ID.

| # define BLOCK\_SIZE xxxx ; *// tells how many bytes are there in one block*  int getIndexBlockNumber ( int fileID );*// takes the file ID and returns the block where index table is stored .*  int \* loadIndexFromBlock ( int blockNumber );*// takes the block number and loads the index table in memory and returns its pointer .*  byte \* loadBytesFromBlock ( int blockNumber ); *// takes the block number and loads raw bytes in that block in memory , and returns its address .* |
| --- |

int getByte ( int logicalByteNumber , int fileID )

{

int logicalBlockNumber = logicalByteNumber / BLOCK\_SIZE ;

int offset = logicalByteNumber % BLOCK\_SIZE ;

int indexNumber = getIndexBlockNumber ( fileID );

int \* index = loadIndexFromBlock ( indexNumber );

byte \* bs = loadBytesFromBlock ( index [ logicalBlockNumber ]) ;

return bs[ offset ];

}

School of Computer Science Page 2 of 3

Question 4 (10 points): Implement the optimal page replacement algorithm using following functions Hint: read the declarations carefully!

| Class List ; *// the same class definition given in Question 1*  Class Iterator ; *// the same class definition given in Question 1*  *// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -* int getNextOccurence ( int pageNumber ); *// returns the position of next occurence of the pageNumber in the reference string .*  List getPageList () ; *// returns the list of all pages loaded in the memory .* |
| --- |

int getPageToReplace ()

{

List l = getPageList () ;

Iterator iter = new Iterator (l);

int pos = 0;

int page = -1;

int pageToReplace = -1;

while (( page = iter . getNext () ) > -1)

{

if ( ( thisPos = getNextOccurence ( page )) > pos )

{

pageToReplace = page ;

pos = thisPos ;

}

}

return pageToReplace ;

}

Question 5 (6 points): List any three conditions which need to be true for a deadlock to occur.

1. 2.

3.

Question 6 (4 points): In deadlock avoidance algorithms, deadlocks are possible structurally, but we keep a gaurd and do not let all those conditions to be true that can result into a deadlock.

1. True 2. False

Question 7 (4 points): In deadlock prevention algorithms, deadlocks are structurally not possible. 1. True 2. False

School of Computer Science Page 3 of 3