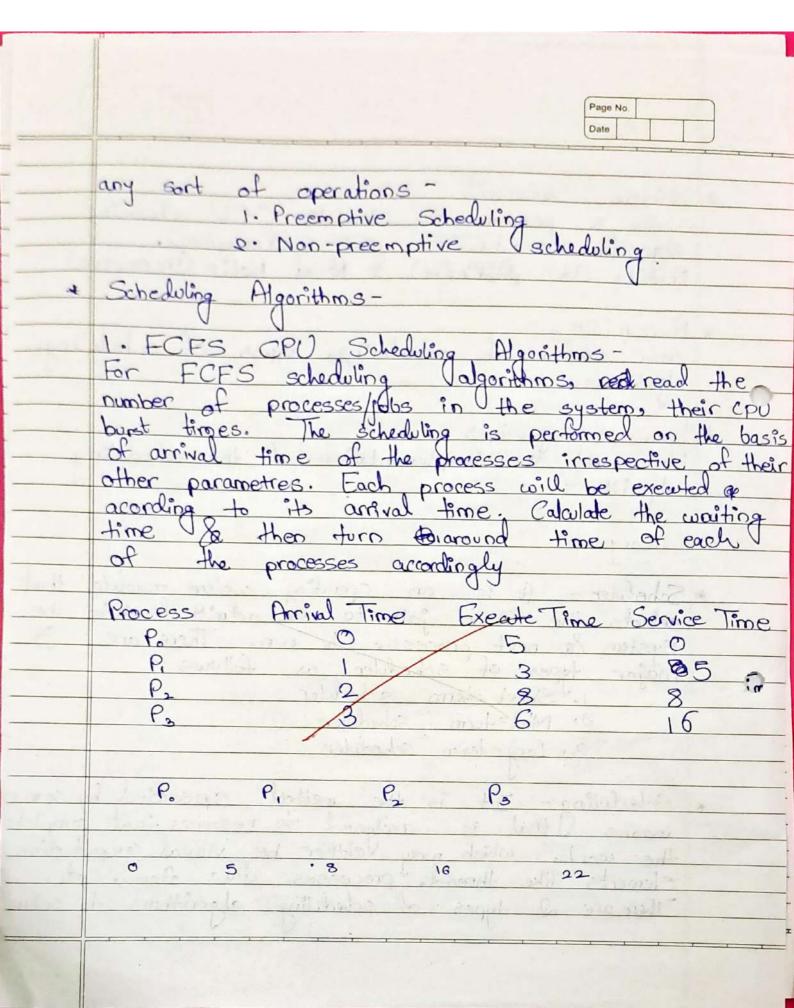


For example, suppose a compliler translates the source language to its target machine language without having the options for generating required because obviously there are some modifications in the compiler itself according to the machine specification. O It machine coole is generated directly from source cooles then for target machine coole, we will have an optimised code generators but if we will have a machine in dependent intermediate code. 1 It is easier to apply source roole modifi-ne - ration to the machine be we see the performance of source code by optimizing the intermediate code. * The following are commonly used intermediate representation -1. Postfix Notation 2. Syntax Tree 3. Three address code * Conclusion - polinger stay Thus, we have studied the data structure & Implementation of Pars I & Pars II of a two-pass macro processor. Because of the harding beautiful

Page No. * Problem Statement -Write a program to simulate CPU adealuling algorithms - FCFS, SJF (Preemptive), J Priority (Non-preemptive) & Round Robin (Preemptive) * trerequisites -Basic concepts of scheduling, types of scheduling, * Learning Objectives -Understand the implementation of the scheduling algorithms * Theory Scheduler - It is an operating system module that selects the next job to be admitted into the system & next process, to run. There are 3 major types of scheduler as follows -1. Short term scheduler 2. Mid=term Scheduler 3. Long-term scheduler Scheduling - It is the method specified by some means I that is assigned to resources that complete the work, which may leither be visual computation clements like thread, processes, data flows, etc.

There are a types of scheduling algorithms to solve

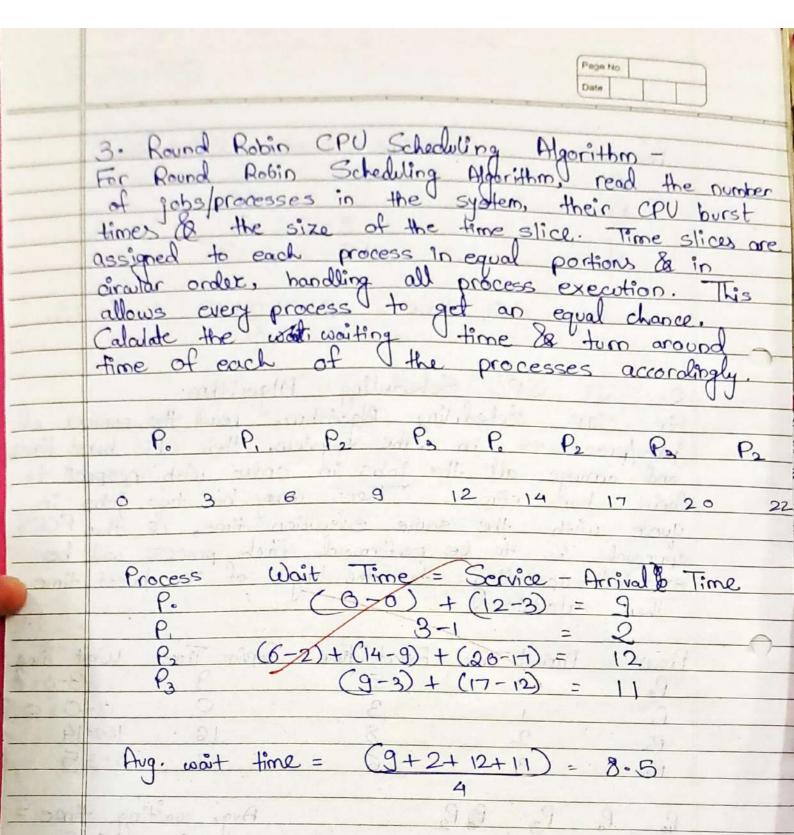


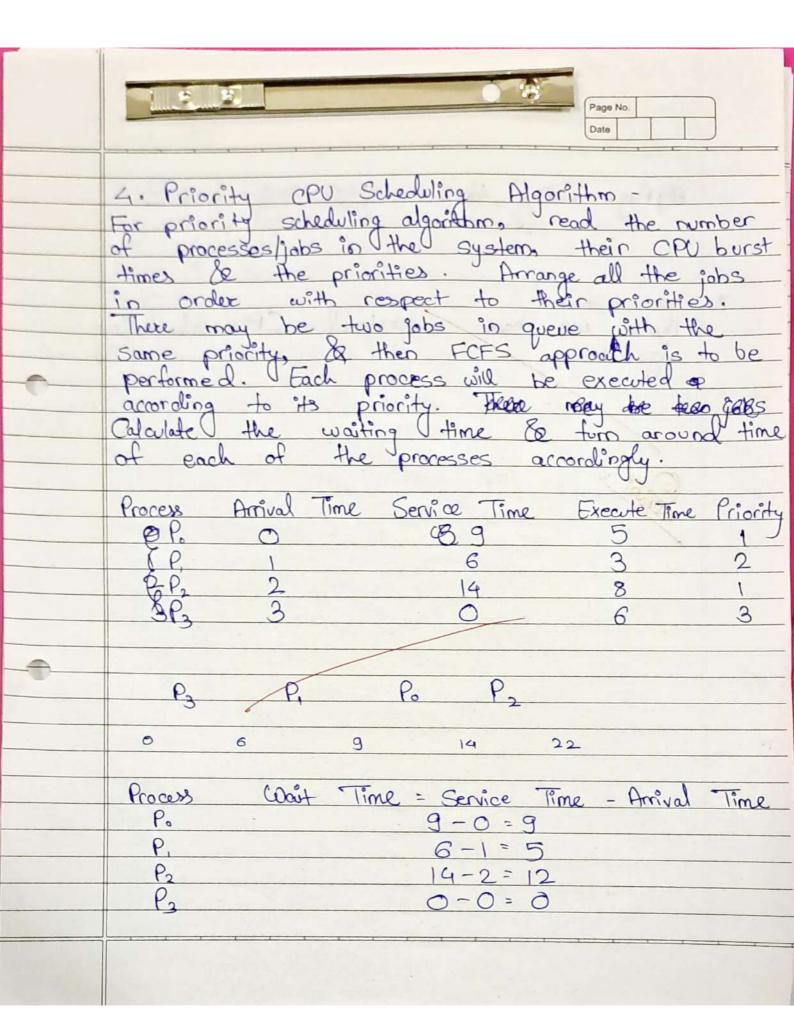
	Page No.
	Date
	Process Wait time = Service - Arrival time
	P. 0-0=0
1	$P_{1} = 4$
	P2 8-2=6
	P3 16-3=13
	.: Avg. wait time = 0+4+6+13 = 5.75.
-	4
	Lucia Carrotti all'il se dell'est de la se
	2. SJF CPU Scheduling Algorithm
	For SJF Scheduling Algorithm, read the number of
	jobs/processes in the Isystem, their CPU burst times.
	and arrange all the jobs in order with respect to
	and arrange all the jobs in order with respect to their burst times. There many be two jobs in
	queve with the same execution time, & then FCFS
	approach is to be performed. Each process will be
	executed according to the length of its burst time.
	Then calculate
	Process Arrival Time Execute Time Service Time Wait Time
	P. 3 3-0=2
	P. 3 0 6-0=0
	P ₂ 2 8 16 16-2=14
	P3 8 -35

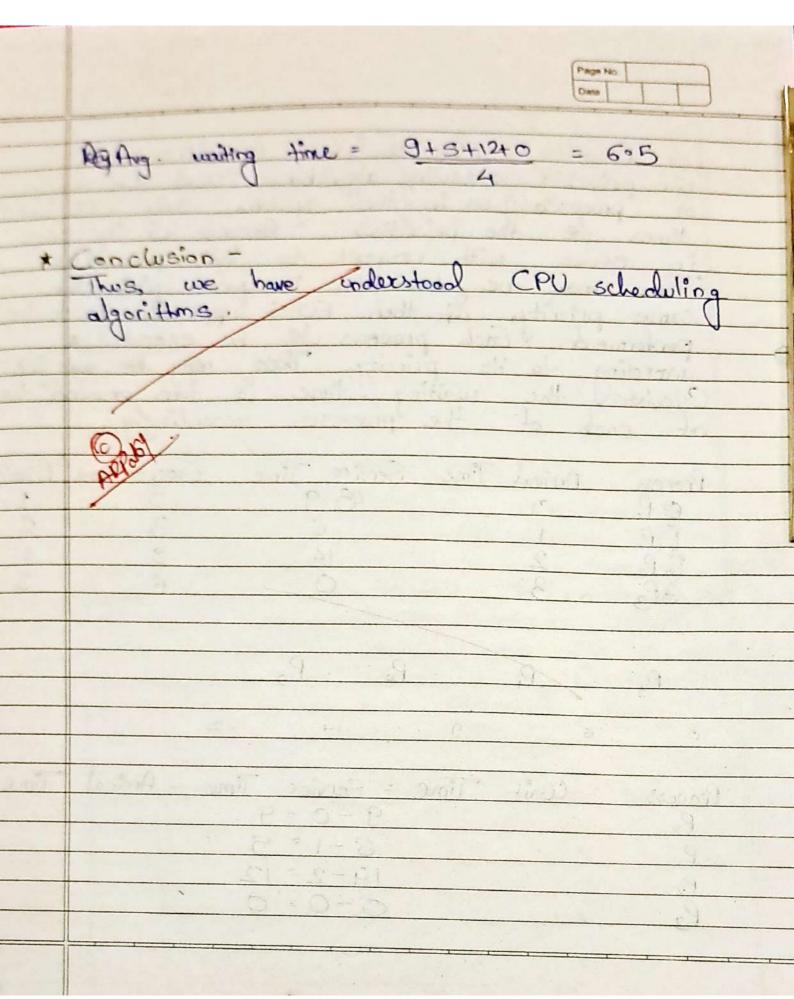
Po Po Pa BP2

8 16 22

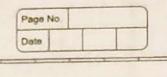
Avg. waiting time = 3+14+5 = 18.5







Page No 40 Date * Problem Statement Write a program to simulate page textellions replacement algorithms - Least Recently Used (LRU) First In First Out (FIFO), Optimal Page Replacement. * Prerequisites - Basic concepts & idea of page replacement. * Learning Objectives
1. To understand page fault in paging
2. To implement various page replacement algorithms. * Theory -In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when a new page comes in * Page Fault -A page fault happens when a running program accesses a memory page that is mapped into the virtual address space but not loaded in physical memory. Since actual physical memory is much smaller than virtual memory, page fault happens. In case of a page fault, operating system might have to replace one



of the existing pages with the newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target of all algorithms is to redwar the number of page faults.

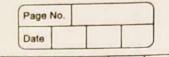
* Page Replacement Algorithms -

First In First Out (FIFO) -- This is the simplest page replacement algorithm. The operating system keeps track of all pages in the memory in a gueve. Here, the delet page is in the front of a gueve. When a page needs to be replaced, page in front of the queve is selected for more removal. -> Eg - Consider page reference string \$1,3, 0, 3, 5, 6, 3 with 3 page frames. Find the number of page faults.

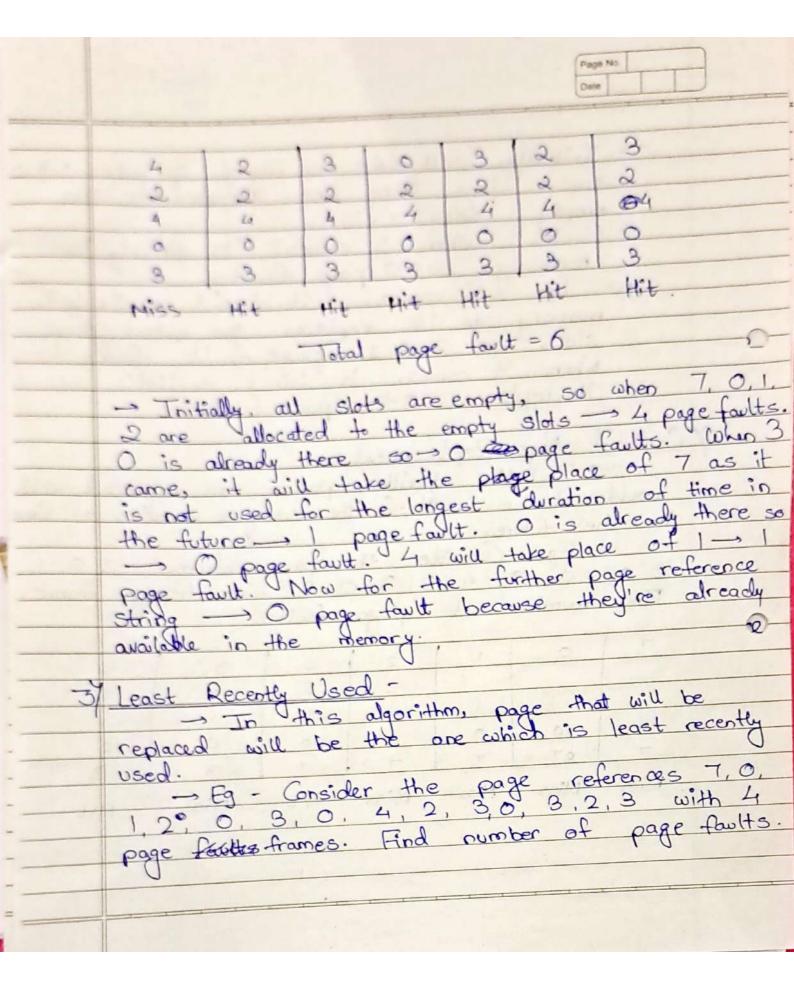
1	3	0	53	5 1	6	. 3
		0	0	0	0	3
THE	3	3	3	3	6	6
1	1	La I said	toold 1	5 5 m	5	5
	1	-	Hit			٠.

Miss

-> Total page faults = B



			1				
	Tritially, all slots are empty, so when 1,3,0 one whom, they are allocated to the empty slots - 3.						
	0	me J	long.	they !	are al	located 1	to the
	e	mpty	slots	-03			
	~ F	aults -	When	3 0	omes,	its alrea	ody in
		memory	, 50	>0 pa	ge fau	lts.	vailable in
	->-	Then	15 a	mes,	o, it i	sn't a	vailable in
	7	nemory	so i	t rep	laces	the olde	est page
		slot, i	·e. 3	$\rightarrow 1$	page	fault.	inally
	0	when i	3 com	es, it	isn	t availa	ble, so
		it repla	rces	0-71.	page +	ault.	-2
27	0-10-1	0	0 1			311	
M	Optimal Page Replacement						
Danis.	would not be used for the longest duration of time						
J. KEVÉ	in the future.						
	-> Eq - Consider the page references 7, 0, 1,						
2 (1)	2,0,3,0,4,2,3;0,3 with 4-page frame.						
4.0	Find number of page faults.						
	- vacana si sa sa sa sa -						
	7	0	1.	2	6	3	0
				2	2	2	2
	A Made 5	sh so	7 100	22 72 3 70 (2)	3144	201-	Y
	r = 3000	0	0	0	0 11	0	1970
	7	07	7	7	7	3	3
	Miss	Miss	Miss	Miss	Hit	Miss	Hit
4.0	444	2 0	longer	1 Lane 1	· Asom A	1.1	200
							.) *



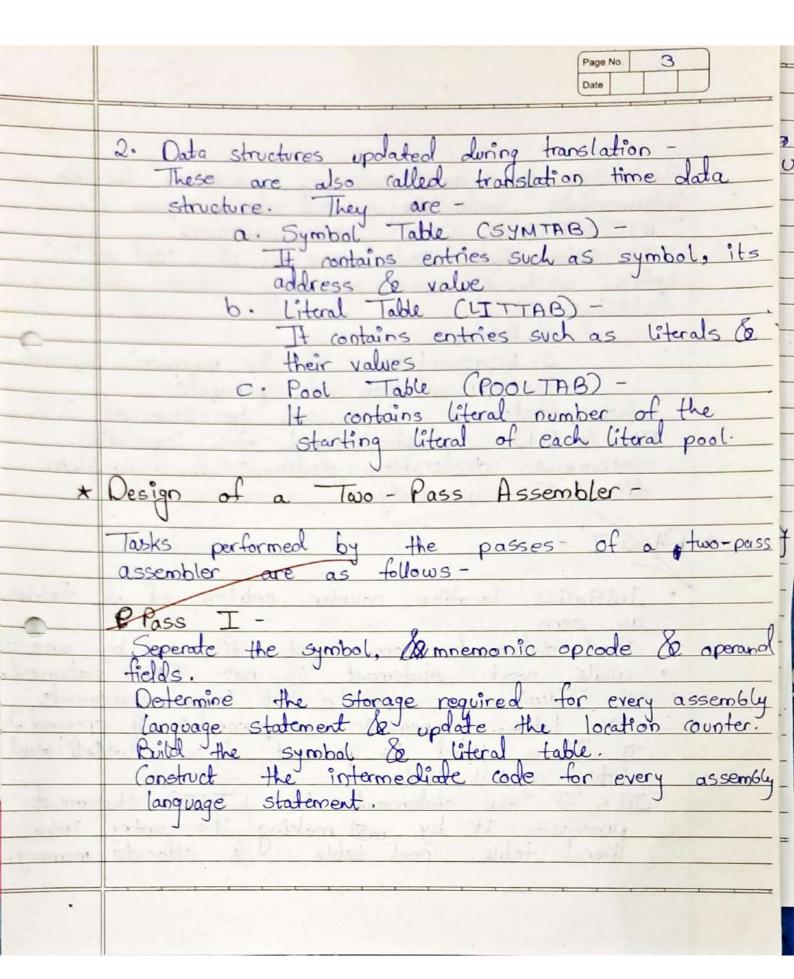
					, (Page No.	
						Date	
	7	0	1	2	0 1	.3.	0
			Die die	2	2	2	2_
				to to a		1	
		0	0	0	0	0	8
	7	4	7	7	7	3	3
	Niss	Miss	Miss	Miss	Hit	Miss	Hit
_	4	2	3	0	3	2	3
	2	2	2	2	2	2	2
	4	4	4	4	4	4	4
	0	0	0	0	0	0	0_
	3	3	3	3	3	3	3
	Miss	Hit	Hit	Hit	Hit	Hit	Hit
			7 4				
	-> Tritic	My. a	U 5/04	s are	empty.	So w	nen T,
	0.1.2	dre	Macates	1 to th	e emoty	slots	\rightarrow
	4 page	faults	. 0	is a	Iready 1	there -	-> 0_
-	4 page	foults.	When	3 0	omes,	t takes	place
	oto 7	as	it is	least	recently	used -	-> l
	map tou	11- (1	10		1 0	$\sim \sim \sim$	0400
	faults.	4	will take	e place	of 1	-> 1 po	ge fault.
	Now, f	or fu	rther	page re	ference,	stringe	2
	O po	ige +	aults	· O bec	ause th	ey and	already
	faults. Now, f O po	J'in	the r	nemory.			
				Ŋ			

Page No. Date * Problem Octinition -Design suitable data structures & implement pays I machine in Java, using Object-oriented feature.

Implementation should consist of a few instructions from each rategory & a few assembler directives. * Prerequisites Basic Concepts of assembler pass I & pass I * Learning Objectives To understand data structures I of an assembler. To implement pass I of an assembler. Theory Concepts -* A language translator bridges an execution gap to machine language of computer system. An assembler is a language translator whose source language is as assembly language. An assembler Vis a program that accepts as input an assembly language program & then converts it into machine * Two - pass translation scheme -

Page No.	2
Date	

In a 2-pass assembler, the first pass constructs an intermediate representation of the source program for use by the second pass. This representation the symbol tables, literal tables, le processed form of the source program, called as intermediate and the symbol tables of variant T. to the entity, which precedes its definition in the program. While processing a statement containing a forward reference, language processor does not possess all relevant information concerning referenced entity. This creates difficulty in synthesizing the equivalent target statements. This problem can be solved by postponing the generation of target rode until more information concerning the entity is available. This recluces memory requirements of RPLP & simplifies its organization. This leads to multi-pass model of language processing. * Data Structures of a Two-Pass Assembler-1. Operation Coole Table (OPTAB) -This is used for storing mnemonic, operation code, and class of instruction.



Date Synthesize the target code by processing the intermediate code dangenerated during pass I.

INTERMEDIATE CODE REPRESENTATION The intermediate code consists of a set of IC units, each IC unit consisting of the following 1. Address 2. Representation of the mnemonic opcode 3. Representation of Operands. Where Statement class can be one of IS, DL & AO, which stand for imperative Statement, declaration statement & assembleur directive respectively * PASS I -Initialize location counter, entries of all tables · Read statements from input file one-by-one. · While next statement is not END statement I. Tokenize or seperate out input statements as lable, mnemonic, or operand 1 & operand 2.
II : If a label is present, insert lateral label into symbol table III. If the statement is LTORGO statement processes it by making its entry into literal table, pool table & allocate memory.

Page No.

