Scheduling Algorithms System

Submitted by:

Mohamed Ahmed Rabea
Mostafa Wael Hussien
Ahmed Sayed Mohamed
Mohamed Yousef Helmy
Ali Abdelmenaim Mohamed

Submitted To:

Dr. Marwa Yousef

Eng. Mohamed Abdelmegid

Contents:

Overview

Overview

The **Process Scheduler Simulator for Multiprocessor Systems** is a comprehensive tool designed to emulate and evaluate different scheduling policies within a multiprocessor environment. The primary objective of this project is to offer a versatile platform that allows users to simulate, analyze, and compare the performance of distinct scheduling algorithms, including First-Come, First-Served (FCFS), Shortest Job First (SJF), Shortest Time-to-Completion First (STCF), Round Robin (RR), Priority, Multilevel Feedback Queue (MLFQ), and Stride scheduling.

Objectives:

- 1. **Scheduling Policy Evaluation:** The simulator facilitates the examination and comparison of diverse scheduling policies to comprehend their behavior in a multi-core processor setting.
- 2. **Performance Assessment:** It measures and reports crucial performance metrics like turnaround time.
- 3. **User Interaction and Customization:** The tool offers an intuitive user interface that enables users to interact with the system, input custom processes, modify parameters, and select scheduling policies for simulation.
- 4. **Visualization and Analysis:** Through Gantt charts, the simulator illustrates the execution and allocation of processes across processors, aiding in the assessment of scheduling policies' efficiency.

Significance:

Understanding the nuances of scheduling policies in a multiprocessor environment is pivotal in optimizing system performance, resource utilization, and responsiveness. This simulator provides an invaluable platform for students, researchers, and system designers to explore.

Scope

The **Process Scheduler Simulator for Multiprocessor Systems** encompasses a wide array of functionalities and capabilities, providing a comprehensive platform for simulating and analyzing various scheduling policies in a multiprocessor environment.

Scheduling Policies Covered:

- 1. **First-Come, First-Served (FCFS):** Processes are executed in the order they arrive.
- 2. Shortest Job First (SJF): Prioritizes the shortest burst time processes.
- 3. **Shortest Time-to-Completion First (STCF):** Selects processes based on remaining time to completion.
- 4. **Round Robin (RR):** Time-sliced scheduling for fair execution among processes.
- 5. **Priority Scheduling:** Assigns priority levels to processes for execution.
- 6. **Multilevel Feedback Queue (MLFQ):** Implements multiple queues with varied priorities.
- 7. **Stride Scheduling:** Utilizes proportional allocation of CPU time based on stride values.

Types of Simulated Processes:

The simulator accommodates a diverse range of processes, comprising both CPU and I/O-bound tasks. Each process is defined by a series of CPU bursts and I/O bursts, mimicking real-world workloads.

User Interaction Capabilities:

Users have an interactive interface that allows for:

- Inputting custom processes with specific burst times and characteristics.
- Modifying system parameters, like quantum size (for RR), Tickets (for Stride).
- Selecting and initiating simulations for different scheduling policies.
- Visualizing results through Gantt charts.

System Architecture

The **Process Scheduler Simulator** exhibits a modular architecture comprising crucial components for emulating a multiprocessor system and implementing various scheduling policies.

Components:

1. Process Representation:

- Defined as a structure encompassing essential attributes like process name, arrival time, phases (execution or I/O), priority, state, tickets, and performance metrics.
- Employs a vector-based representation for each process, allowing flexible management and manipulation during simulation.

2. Scheduler Module:

- Core component responsible for executing the Scheduling policies.
- Houses algorithms and logic for scheduling decisions based on the chosen policy.
- Integrates with process queues, handles context switching, and manages process states.

3. Multiprocessor Environment:

- Simulates a four-processor system, utilizing an array-based representation to manage processor states and process assignments.
- Facilitates parallel execution of processes based on availability and task requirements.

4. User Interface:

- Interaction primarily via the command-line interface for simplicity.
- Provides user prompts to input the number of processes and generates sample processes with randomized attributes within a file and the system takes the input from the file itself.
- Initiates the Scheduler chosen policy and displays the output, including processor-wise execution snapshots.

High-Level Workflow:

1. Initialization and Input:

- Accepts user input for the number of processes or generates sample processes with randomized attributes from the file.
- Initializes process structures and computes the necessary values for each process based on the provided phases and arrival times.

2. Scheduler Execution:

- Utilizes a loop-based mechanism to simulate processor allocation and execution of processes based on Scheduling.
- Manages CPU bursts, I/O phases, context switches, and process completions.

3. Output and Analysis:

- Generates processor-wise execution snapshots using a twodimensional vector (output) to represent the progress of each process over time.
- Calculates process statistics such as turnaround time and completion time for each process.

Scalability and Extension:

The current architecture is tailored specifically for the Scheduling policies within a four-processor environment. Extending this simulator to accommodate additional scheduling policies or processors would require structural modifications and enhanced user interaction capabilities.

Data Structures

1. Process Structure (struct process):

- Attributes: process_name, arrive_time, turn_around_time, complete_time, time_consumed, n_phases, phase_idx, last_processor, phases, state, priority, current_brust_time, pass_value, stride, tickets, qunta.
- Represents an individual process with its specific characteristics, including name, arrival time, phases, state, and performance metrics.
- Utilizes a vector to store phase details (execution time and type CPU burst or I/O burst).

2. Output Matrix (vector<vector<string>> output):

- A two-dimensional vector used to visualize the execution progress of processes on each processor over time.
- Provides a snapshot of process execution in a tabular format, aiding in creating Gantt charts or processor-wise execution logs.

3. Scheduler Data Structures:

- queue<int> for handling waiting processes during execution.
- Arrays (processor[], WillGoToTheQ[]) to manage process allocation to processors based on the Stride Scheduling policy.

Algorithms Implementation

1. Process Initialization and Input Generation:

• **Mechanism:** Generates or accepts input for the number of processes, their attributes, and phases from a file.

• Working Principle:

- Generates sample processes with randomized attributes for testing and simulation purposes.
- Accepts user input for process attributes like arrival time, phases (CPU bursts or I/O), and other relevant parameters.
- Computes necessary values for each process based on provided phase details for each scheduling policy.

Integration into Simulator:

- Initializes process structures and sets up the simulation environment with sample or user-defined input from a file.
- Enables the execution of scheduling algorithms by creating a pool of processes for scheduling.

2. SJF Algorithm Implementation

1. Initialization:

- Sorts the processes based on their burst times using the comparedByBurst comparison function.
- Initializes variables like finished_processes, processor[],
 WillGoToTheQ[], and a priority queue waiting to manage waiting processes.

2. Scheduler Execution:

- Loops through each time unit to simulate the scheduler's behavior.
- Checks for arriving processes at each time unit and allocates available processors to the shortest job among the arrived processes.
- Queues processes that arrive but cannot be immediately executed due to all processors being busy.

- Executes processes on the processors based on their burst times, allowing the shortest job to run on the available processor.
- Tracks the execution progress in the output matrix for visual representation.
- Handles completion of phases, updating states, and marking processes as finished when all phases are executed.

3. Processor Allocation and Execution:

- Assigns available processors to the shortest job among the arrived processes.
- Manages the execution of processes based on their burst times, ensuring completion of phases, and marking processes as finished accordingly.

4. IO handler bypg Function:

• Handles I/O operations for processes in the priority queue waiting, similar to the IO handler function used in other scheduling policies.

5. Completion Check:

• Checks if all processes have finished execution. If so, generates the final output and exits the simulation.

3. FCFS Algorithm Implementation

1. Initialization:

Initializes variables such as finished_processes, processor[],
 WillGoToTheQ[], and a queue waiting to manage waiting processes.

2. Scheduler Execution:

- Loops through each time unit to simulate the scheduler's behavior.
- Checks for arriving processes at each time unit and allocates available processors to the arriving processes based on the FCFS principle.
- Queues processes that arrive but cannot be immediately executed due to all processors being busy.

3. Processor Allocation and Execution:

- Allocates processors to arriving processes in the order of their arrival.
- Manages the execution of processes on processors, allowing them to execute their phases according to their burst times.
- Tracks the execution progress in the output matrix for visual representation.

4. IO handler Function:

• Handles I/O operations for processes in the waiting queue, similar to other scheduling policies.

5. Completion Check:

• Checks if all processes have finished execution. If so, generates the final output and exits the simulation.

4. STCF (Shortest Time-to-Completion First) Algorithm Implementation

1. Initialization:

Initializes variables like finished_processes, processor[],
 WillGoToTheQ[], and a priority queue waiting to manage waiting processes based on their burst times.

2. Scheduler Execution:

- Loops through each time unit to simulate the scheduler's behavior.
- Checks for arriving processes at each time unit and allocates available processors based on the STCF principle, favoring processes with shorter burst times.

3. Processor Allocation and Execution:

- Allocates processors to arriving processes based on their burst times.
- Manages the execution of processes on processors, allowing them to execute their phases according to their burst times.
- Tracks the execution progress in the **output** matrix for visualization.

4. IO_handler_bypq Function:

• Handles I/O operations for processes in the waiting queue, similar to other scheduling policies.

5. Completion Check:

• Checks if all processes have finished execution. If so, generates the final output and exits the simulation.

5. Round Robin Algorithm Implementation

1. Initialization:

Initializes variables like my_queue (queue to store processes),
 finished_processes, processor[], WillGoToTheQ[], and time_slice (quantum time) for process execution.

2. Scheduler Execution:

- Simulates the scheduler's behavior by iterating over each time unit.
- Checks for arriving processes and assigns them to available processors or enqueues them in the **my_queue**.

3. Processor Allocation and Execution:

- Allocates processors to arriving processes or those waiting in the queue.
- Executes processes for a fixed time slice (**time_slice**) or until the process finishes its current burst time, whichever occurs first.
- Tracks the progress of process execution in the output matrix for visualization.

4. Match Preferences and IO Handling:

- Matches preferences for processor allocation.
- Handles I/O operations for processes.

5. Completion Check:

• Checks if all processes have finished execution. If so, generates the final output and exits the simulation.

6. Multi-Level Feedback Queue (MLFQ) Algorithm Implementation:

1. Initialization:

- Initializes variables like my_queue[] (queues for multiple priority levels), finised_processes, processor[], WillGoToTheQ[], time_slice, and other related parameters.
- Resets the processors and queues.

2. Scheduler Execution:

- Iterates over each time unit.
- Assigns arriving processes to available processors or to appropriate queues based on priority levels (my_queue).

3. Processor Allocation and Execution:

- Executes processes in processors based on their priority levels.
- Tracks the progress of process execution in the output matrix for visualization.
- Handles time quantum expiry and phase completion for processes, moving them to the appropriate queue or updating their priorities.

4. Queue Management and Priority Handling:

- Manages queues by shifting processes between different priority queues based on their quantum usage and priorities.
- Handles processor allocation considering process priorities and available queues.

5. Match Preferences and IO Handling:

- Matches preferences for processor allocation.
- Handles I/O operations for processes.

6. Priority Adjustment:

 Adjusts the priority of processes after a certain number of time units (time_to_moveup), likely to maintain fairness or adjust scheduling parameters dynamically.

7. Completion Check:

• Checks if all processes have finished execution. If so, generates the final output and exits the simulation.

7. Stride Scheduling Algorithm Implementation:

1. Initialization:

Initializes variables such as waiting, q, finished_processes,
 WillGoToTheQ, processor[], and i.

2. Scheduler Execution:

- Executes a loop representing each time unit.
- Sorts the processes based on their arrival time or their pass value and fills the queue accordingly.
- Assigns arriving processes to available processors or to the waiting queue (waiting).

3. Processor Allocation and Execution:

- Executes processes in processors based on their arrival times or pass values.
- Tracks the progress of process execution in the output matrix for visualization.
- Updates the pass value for each process based on its stride.
- Handles completion of processes after their time slice or phase completion.

4. Queue Management and Preference Handling:

- Manages the waiting queue and assigns processes to processors based on their states and availability.
- Matches preferences for processor allocation.

5. **IO Handling and Completion Check:**

- Handles I/O operations for processes.
- Checks if all processes have completed execution. If so, generates the final output and exits the simulation.

6. Completion Check:

• Checks if all processes have finished execution. If so, generates the final output and exits the simulation.

8. Priority Scheduling Algorithm:

1. Initialization:

- Sorts the processes based on priority.
- Initializes variables like finished_processes, processor[],
 WillGoToTheQ[], and queues.

2. Scheduler Execution:

- Executes a loop representing each time unit.
- Assigns arriving processes to available processors based on their priority.
- Allocates processes that arrive when no processors are available to a waiting queue.

3. Processor Allocation and Execution:

- Executes processes in processors based on their priority.
- Tracks the progress of process execution in the output matrix for visualization.
- Handles the completion of processes after their time slice or phase completion.

4. Queue Management and Preference Handling:

- Manages the waiting queue and assigns processes to processors based on their states and availability.
- Matches preferences for processor allocation using the WillGoToTheQ array.

5. **IO Handling and Completion Check:**

- Handles I/O operations for processes.
- Checks if all processes have completed execution. If so, generates the final output and exits the simulation.

Code Samples:

Round Robin

20 EXE

1 1/0

В 3 О

6 EXE

2 I/O

1 EXE

C 3 0

10 EXE

2 1/0

10 EXE

D 3 0

6 EXE

3 I/O

6 EXE

E 3 0

6 EXE

3 I/O

6 EXE

F 3 0

6 EXE

3 I/O

6 EXE

X 3 2

1 EXE

1 1/0

10 EXE

average turn around time = 19.1429	turn	C arrived at 0 it's turn arround E arrived at 0 it's turn arround	A arrived at 0 it's turn arround time = 29 B arrived at 0 it's turn arround time = 9	·····processes	processor 4 : D 0 D 1 D 2 D 3 D 4 D 5 X 6 A 7 A 8 A 9 A 18 t 25 t 26 t 27 t 28 t	processor 3 : C 0 C 1 C 2 C 3 C 4 C 5 C 6 C 7 C 8 C 9 D 10 t 25 t 26 t 27 t 28 t
129	d time = 22 d time = 17	time = 21 d time = 15 d time = 21	time = 29		2 0 3 0 4	2 C 3 C 4
					D 5 X 6 A	C 5 C 6 C
					7 A 8	17 018
					A 9 A 10	C 9 D 10

SJF	× \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \	APT	, bi	, bu	CP
7 A 4 0	arrived arrived arrived arrived arrived arrived arrived	processor A 25 {	processor i 25 i	processor i 25 i	processor C 25 C
		4 —	— 3	- 2	— H
3 EXE	0 1t 0 1t 0 1t 0 1t 0 1t 1	: E 26 t	: D 1	: B 1	: A 1
2 1/0	::	0 E	0 0	0 8	0 A
		1 E	1 0	1 B	1 A
1 1/0		2	2	2	2
	time = time = time = time = time =	E 3	D 3	B 3	x 3
B 3 0	25 10 16 16 21 21 27 13	E	0 -	B _	71
6 EXE		4 E	4 D	4 B	4 F
2 1/0		- 5	5	- 5	- 5
1 EXE		A 6	× – 6	c 6	F 6
		A -	×	c	F 7
C30		7 A	7 X	7 C	
10 EXE		-8	- 8	- 8	F 8
2 1/0		A 9	x 9	6 1 3	F 9
10 EXE		<u>^</u>	<u>×</u>	<u>c </u>	B —
IO EXE		10 A	10 X	10 C	10 D
		111	- 11	- ::	11
D 3 0		Α -	×	<u>c </u>	0
6 EXE		12 A	12 X	12 C	12
3 1/0		1 13	(13	: 13	D 13
6 EXE		A -	×	C	D
		14 A	14 X	14 C	14 0
E 3 0		15	15	15	D 15
6 EXE		A _	F	E 16	D
3 I/O		16 A	16 F	16 E	16 t
6 EXE		\ 17	17	17	1 17
		A -	-	m	0
F30		18 A	18 F	18 E	18 0
6 EXE		\ 19	: 19	19	C 19
3 I/O		<u>^</u>	F	E 20	C 20
6 EXE		20 A	20 F		
		A 21	F 21	E 21	C 21
X 3 2		A	=	l i 22	l C 22
1 EXE		22	22		
1 I/O		A 23	t 23	۱ 23	C 23
10 EXE		8 A I	1 2	3 t 24	100
		24	24	24	C 24

FCFS	، به	× 7 E D C B A +	סס	- P	40	סד
7	аvегаде	arrived arrived arrived arrived arrived arrived	processor A 25 A	processor	processor i 25 i	processor F 25 i
A 4 0	turn	ed at red	4	w	N	-
3 EXE	around	2000001	: D	: 0	: B	: A 0
2 I/O	nd ti		0 D	0 0	0 B	
20 EXE	me .		1	1	1	A 1
11/0	19.1429	ocesses. arround arround arround arround arround	D 2	C 2	B 2	A 2
	29	time time time time	2 D _	C_	2 в –	2 E
B30		= 26 = 9 = 21 = 15 = 21 = 25 = 17	3 D	3 C	3 B	3 E
6 EXE			_ _	4	4	
2 I/O			0	<u>c</u>	В	E _
1 EXE			5 ×	5 C	5 F	5 E
			6	6	6	9
C30			A 7	c 7	F 7	E 7
			A .	0	F	E 8
LO EXE			8 A	8 C	8 F	8 B
21/0			9	- 9	9	9
0 EXE			A	0	<u>-</u>	X 10
			10 A	10 D	10 F	
30			11	1 11	_ =	X 11
EXE			A	0	- C	
1/0			12	12	12	X 12
EXE			A 1	D 1	c 1	× 1
			13 A	13 D	13 C	13 X
3 0			14	14	14	14
EXE			A	0	C	X 15
1/0			15 A	15 E	15 (
EXE			1 16	16	C 16	X 16
			A -	E	0 0	×
0			17	17	17	x 1/
XE			A —	E	C 1	X 18
/0			18 A	18 E	18 C	8 ×
EXE			19	19	19	X 19
			A -	<u>=</u>	C 20	F 20
3 2			20 A	20		
EXE			\ 21	E 21	C 21	F 21
1/0						
10 EXE			A 22	t 22	t 22	F 22
			A 2	ί 2	٤ 23	F 23
			23 A	23 i	23 l	
			24	24	24	F 24

MLFQ	o)	· × 11 11 11 11 11 11 11 11 11 11 11 11 1	0.5	d P	A P	d-P
7	average	arrived arrived arrived arrived arrived arrived arrived	processor C 25 C	processor i 25 i	processor A 25 A	processor
A 4 0	e turn	ved a a ved a	sor 4 C	sor 3	SOF 2 A	sor 1
3 EXE	n around	4004	: D 26 C	: c	: B 26 A	: A
2 1/0			0 D	0 C	0 B	0 A
20 EXE	time =	turn ar turn ar turn ar turn ar turn ar	7 1	1 1	3 1 A	7 1
1 1/0	19.1429	arround arround arround arround arround arround arround	F -	E	B	<u>×</u>
	129	time time time time time	2 F	2 E	2 B	2 C
B30		28 = 28 0 = 10 0 = 27 0 = 18 0 = 19 0 = 19	- 3	3	- 3	C 3
6 EXE			D 4	× – 4	B 4	C 4
2 1/0			0	A	8	<u>c</u>
1 EXE			5 D	5 E	5	UI.
			- 6	- 6	- 6	c 6
C30			D	E _	-	x 7
10 EXE			7 A	7 E	7 F	2000
2 1/0			-8	-8	8	× – 8
10 EXE			A 9	E 9	F 9	6 x
			Α	c _	8	× _
D30			10	10	10	10
6 EXE			D 1	c 1	A 11	X 11
			11 0	11 C	1 A	
3 1/0			12	12	12	X 12
6 EXE			A	<u>m</u>	F —	X 13
			13 A	13 E	13 F	
E 3 0			14	14	14	X 14
6 EXE			A .	c l	0	X 15
3 I/O			15	15	15	
6 EXE			A 16	E 16	D 16	F 16
			6 A	m	6 D	
F 3 0			17	17	D 17	F 17
6 EXE			<u>c </u>	E 18	D 18	F 18
3 I/O			18 C			
6 EXE			19	E 19	A 19	F 19
			<u>c</u>	_	A 20	t 20
X 3 2			20 (20	20	
1 EXE			C 21	t 21	A 21	t 21
1 1/0			1 (1 1	1 A	
10 EXE			22	22	A 22	í 22
			<u>c </u>	-	A 23	t 23
			23 C	23 t		
			24	24	A 24	t 24

STRIDE	
7	A Proc
A 4 0	processor 3 : C 0 i 25 i 26 i 0 i 25 i 26 i 0
3 EXE	3 : C 3 : C 4 : D 1 26 at 0 to at
2 1/0	
20 EXE	TO A
1 I/O	N N N
	C
B 3 0	C D D 26 118 115 117
6 EXE	4 C
2 1/0	5 S
1 EXE	_ 6 A
	7 7 1 7
C30	A C 8
10 EXE	A C 9
2 1/0	A 0
10 EXE	10 D
D 3 0	D 12 A 12
6 EXE	2 D
3 I/O	13 D
6 EXE	14
	D 15 A 15
E 3 0	> 11
6 EXE	16 16
3 1/0	F 17 A 17
6 EXE	A F 18
	> π
F30	and the second s
6 EXE	A 20 A 20
3 1/0	F 21 A 21
6 EXE	
V 2 2	22 22
X 3 2 1 EXE	A (23
11/0	<u> </u>
10 EXE	2 24
TO LAL	

processor 2:8 | 0 8 | 1 8 | 2 8 | 3 8 | 4 8 | 5 F | 6 F | 7 F | 8 F | 9 F | 10 F | 11 E | 12 E | 13 E | 14 E | 15 E | 16 E | 17 B | 18 t | 19 t | 20 t | 21 t | 22 t | 23 t | 24 t | 25 t | 26 t | 27 t

processor 1: A | 0 A | 1 A | 2 E | 3 E | 4 E | 5 E | 6 E | 7 E | 8 X | 9 X | 10 X | 11 X | 12 X | 13 X | 14 X | 15 X | 16 X | 17 X | 18 C | 19 C | 20 C | 21 C | 22 C | 23 C | 24 C | 25 C | 26 C | 27 C

STCF		· X C T E O B A ·		0.7		
7	averag	9 9 9 9 9	processor	processor C 25 C	process	processor A 25 A
A 4 0	e tur	rrived a	sor 4	sor 3	sor 2	SOF 1
3 EXE	n arou	7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	: E	: B 26 C	: D 26 1	: A 26 A
21/0	und ti		0 E	0 B	0 D	0 A
20 EXE	me =	turn ar turn ar turn ar turn ar turn ar	1	<u>-</u> 1		A 1
	18.714	ocesses arround arround arround arround arround	X 2 28 t	8 2 28 i	0 2 28 t	A 2 28 A
11/0	43	time time time time	2 E	2 B	2 D	2 F 1 29
		= 34 = 8 = 14 = 15 = 20 = 27 = 13	3 E	i	3 D	3 A F
B 3 0			30 .	30	30	30 /
6 EXE			[] 31	B 5	D :	F 5 A 31
2 1/0		1	- E	10	-×	A F
1 EXE			6 A	6 C	6 X	6 F
			1 7	7 t 33	7 1 33	7 A 33
C30			3 (8 8 3 i	÷×	~ <u>-</u>
10 EXE			8 C	8 A	8 D	× .
2 1/0			9 E	9 (9 0	9 ×
10 EXE			10	10	10	10
			E -	0	0	×
D30			11 E	11 C	11 D	11
6 EXE			12	: 12) 12	X 12
			E	C I	0	×
31/0			13	13	13	13
6 EXE			E 14	C 14	D 14	X 14
			E -	10	-	
E 3 0			15	15	15	X 15
6 EXE			ί 16	A 16	F 16	t 16
3 I/O			6 i		6 F	
6 EXE			17	A 17	17	ί 17
			ί 18	C 18	F 18	A 18
F 3 0			-	8 C	8 F	
6 EXE			19	19	19	A 19
3 I/O			i 20	C 20	F 20	A 2
6 EXE			0 i	C	~	A 20 A 21
			21	21	21	
X 3 2			ί 2	C 2	ί 2	A 22
			22 i	22 C	22 t	
1 EXE			23	23	23	A 23
11/0			<u>-</u>	<u>c </u> ;	1 1	A 24
10 EXE			24	24	24	24

Priority processor A 4 0 3 EXE B 27 2 1/0 **20 EXE** D 28 8 28 C | 1 1/0 C | 3 | 29 1 = 14 = 21 = 8 = 8 = 14 = 34 = 20 3 C | 4 1 | 30 B 3 0 30 30 t C | 5 _ 0 _ B 6 EXE | 5 | 31 | 5 2 1/0 C | 6 1 | 32 32 32 1 EXE C | 7 C C30 **10 EXE** 2 1/0 **10 EXE** D 3 0 6 EXE 3 I/O 6 EXE E 3 0 6 EXE 3 1/0 6 EXE F 3 0 6 EXE 3 I/O 20 6 EXE X 3 2 1 EXE 1 1/0 **10 EXE**

E | 28

30

31

32

A | 7 A A | 33

