Project Title: OS Game System – A Web-Based Operating System Simulator

1. Introduction:

The OS Game System is a web-based educational platform developed to support the practical understanding of core Operating System (OS) concepts. The system provides interactive simulations of deadlock detection, memory allocation, CPU scheduling, and an integrated quiz module.

Operating Systems is considered one of the most challenging subjects in Computer Science due to its abstract nature and complex theoretical concepts. Traditional learning methods rely mainly on textbooks and lectures, which often fail to provide hands-on experience. As a result, students struggle to visualize system behavior such as resource sharing, scheduling order, and memory utilization.

Previous tools and simulators primarily focus on theoretical explanations or command-line based implementations. These systems lack user-friendly graphical interfaces and real-time interaction, making them less effective for beginners.

This project integrates multiple OS concepts into a single browser-based platform. The system architecture and user interface were implemented using HTML and CSS, while all algorithmic logic was developed using JavaScript. The project provides an intuitive and interactive environment that allows students to experiment with OS algorithms and immediately observe the outcomes.

2. Problem Statement

Operating Systems is one of the most conceptually difficult subjects for computer science students. Many learners face difficulties understanding complex mechanisms such as deadlock, memory management, and CPU scheduling through theory alone.

Limitations of Existing Systems

- ► Lack of visualization tools
- ► No real-time user interaction
- ► Absence of integrated simulation platforms
- ► Command-line based tools are not beginner friendly
- ► Limited learning engagement

Research Gaps Identified

- ► Need for web-based OS simulators
- ► Lack of combined modules in a single platform
- ► Absence of student-focused UI design
- ► Lack of interactive learning tools

The OS Game System addresses these limitations by providing a browser-based simulator with a graphical interface. It integrates multiple OS modules into one platform, enabling users to test algorithms, visualize results, and gain practical knowledge.

3. Objectives

The main objectives of this project are:

- ► To simulate deadlock detection mechanisms
- ► To demonstrate memory allocation algorithms
- ► To implement CPU scheduling strategies
- ► To provide interactive learning through quizzes
- ► To improve conceptual understanding of OS
- ► To support academic laboratory activities
- ► To make OS learning engaging and practical

4. Application

Real-World Use Cases

- ► University OS laboratory demonstrations
- ► Classroom teaching support
- ► Student self-practice tool
- ► Exam preparation and revision

Industry / Academic Relevance

- ►Helps students understand OS internals
- ►Useful for training and research
- ►Supports academic coursework

Potential Users

- ► Undergraduate students
- ► Instructors
- ► Self-learners

Operational Scenarios

- ► Testing deadlock situations
- ► Comparing CPU scheduling algorithms
- ► Practicing memory allocation strategies
- ► Conducting quiz assessments

5. Technologies Used

- ► HTML5
- ► CSS3
- ► Vanilla JavaScript
- ► Visual Studio Code
- ► GitHub

6. System Modules and Functionalities

The OS Game System consists of four main modules:

6.1 OS Quiz Module

Purpose
This module is designed to test and reinforce theoretical knowledge of Operating Systems through interactive multiple-choice questions.

Features

- ►Contains 10 MCQ questions
- ►Covers core OS topics such as:
  - ⇒Deadlock
  - ⇒Memory management
  - ⇒Scheduling
  - ⇒Virtual memory
- ► Life-based scoring system
- ► Real-time score display
- ► Automatic reset after completion

Functionality
Users select answers from given options. Correct answers increase the score, while incorrect answers reduce lives. The final score is displayed after completing the quiz.

6.2 Deadlock Detection Module

Purpose
This module simulates deadlock detection using Allocation, Request, and Available matrices.

Features

- ► Accepts user input matrices
- ► Implements Banker-style detection logic
- ► Detects circular wait condition
- ► Displays exact deadlocked processes
- ► Supports safe and unsafe states

Functionality

The system checks whether each process can complete execution based on available resources. If some processes remain unfinished, the system identifies them as deadlocked.

6.3 Memory Allocation Module

Purpose

Demonstrates memory management using dynamic partitioning strategies.

Algorithms Implemented

- ► First Fit
- ► Best Fit
- ► Worst Fit

Features

- ► Displays allocated memory blocks
- ► Shows remaining memory
- ► Handles:
  - ⇒Exact fit cases
  - ⇒External fragmentation
  - ⇒Insufficient memory
- ► Allows comparison of strategies

Functionality
Users enter memory block sizes and process requirements. The system applies each algorithm and displays allocation results.

6.4 CPU Scheduling Module

Purpose
Simulates process scheduling techniques used by operating systems.

Algorithms Implemented
- ► First Come First Serve (FCFS)
- ► Shortest Job First (SJF)
- ► Round Robin
- ► Priority Scheduling

Features
- ►User input for:
  - ⇒Burst time
  - ⇒Priority
  - ⇒Time quantum
- ►Displays execution order
- ►Shows completion time for each process

Functionality
The system calculates scheduling order based on selected algorithms and displays completion times.

7. Methodology

Step 1: Planning

- ► Defined project scope
- ► Selected OS topics
- ► Designed module structure

Step 2: System Design

- ► Created UI wireframe
- ► Designed navigation flow
- ► Planned algorithm integration

Step 3: Development

- ► Developed UI using HTML and CSS
- ► Implemented logic using JavaScript
- ► Developed modules:
  - ⇒Quiz
  - ⇒Deadlock detection
  - ⇒Memory allocation
  - ⇒CPU scheduling

Step 4: Implementation

- ► Integrated all modules
- ► Connected UI buttons
- ► Displayed outputs dynamically

Step 5: Testing

- ►Tested positive and negative cases
- ►Checked incorrect input handling
- ►Verified accuracy

Step 6: Deployment

- ►Uploaded project to GitHub
- ►Verified browser compatibility

8. Results

The system meets all stated objectives successfully.

Key Outcomes

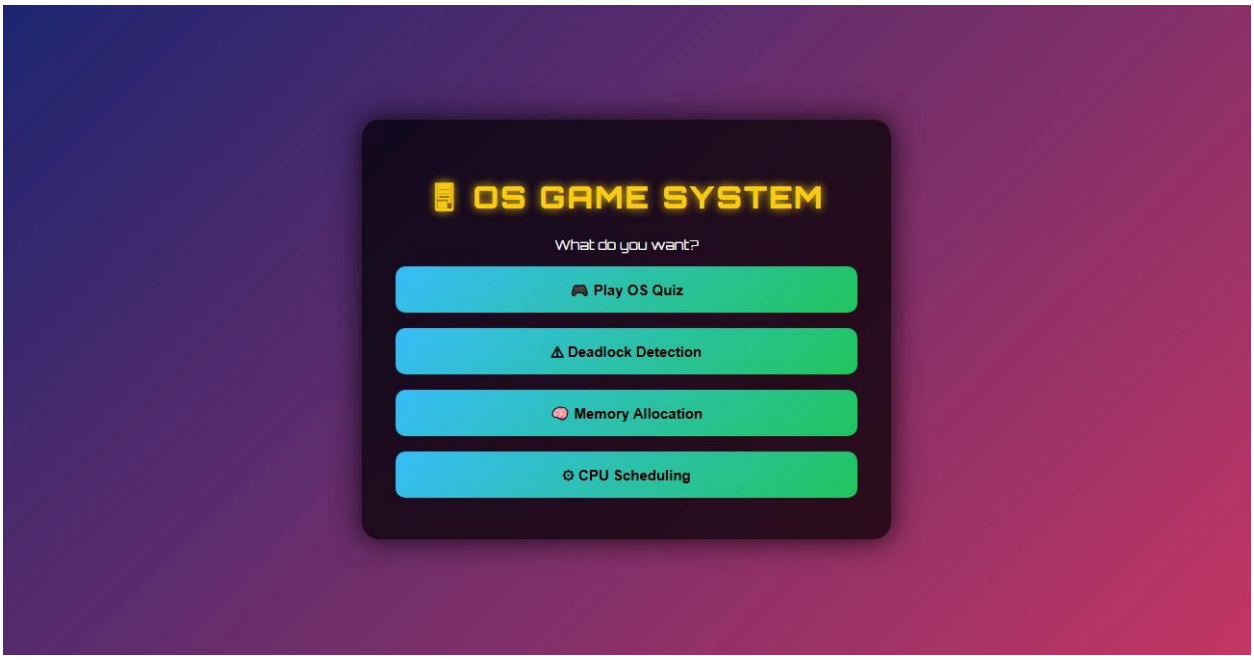
- ►Accurate deadlock detection
- ►Correct memory allocation results
- ►Proper CPU scheduling execution
- ►Functional quiz system

Performance

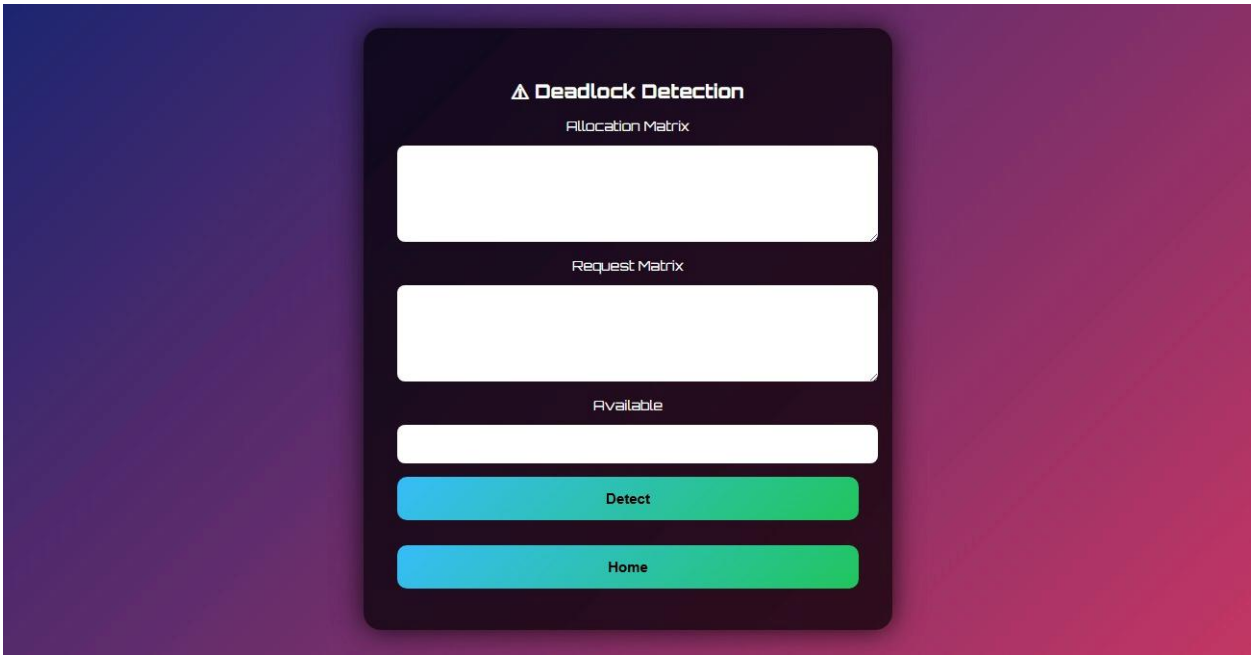
- ►Instant response
- ►No page reload
- ►Works on all modern browsers
- ►The system produces correct outputs for all test cases.

9. Screenshots

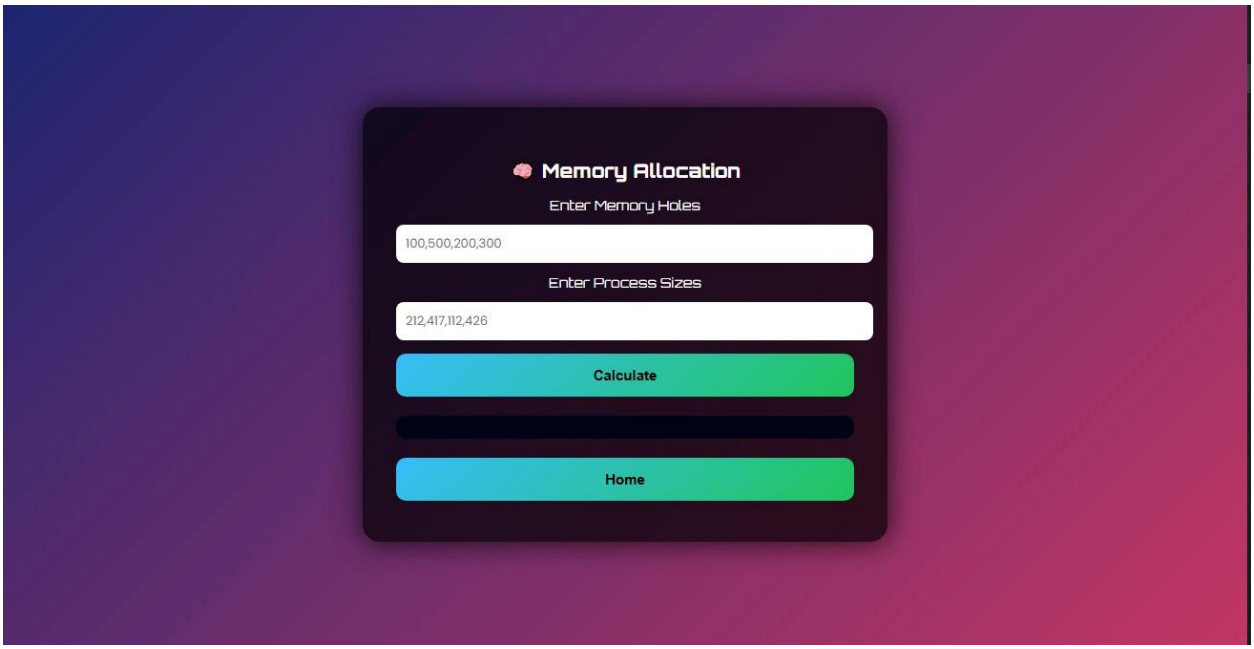
Home page



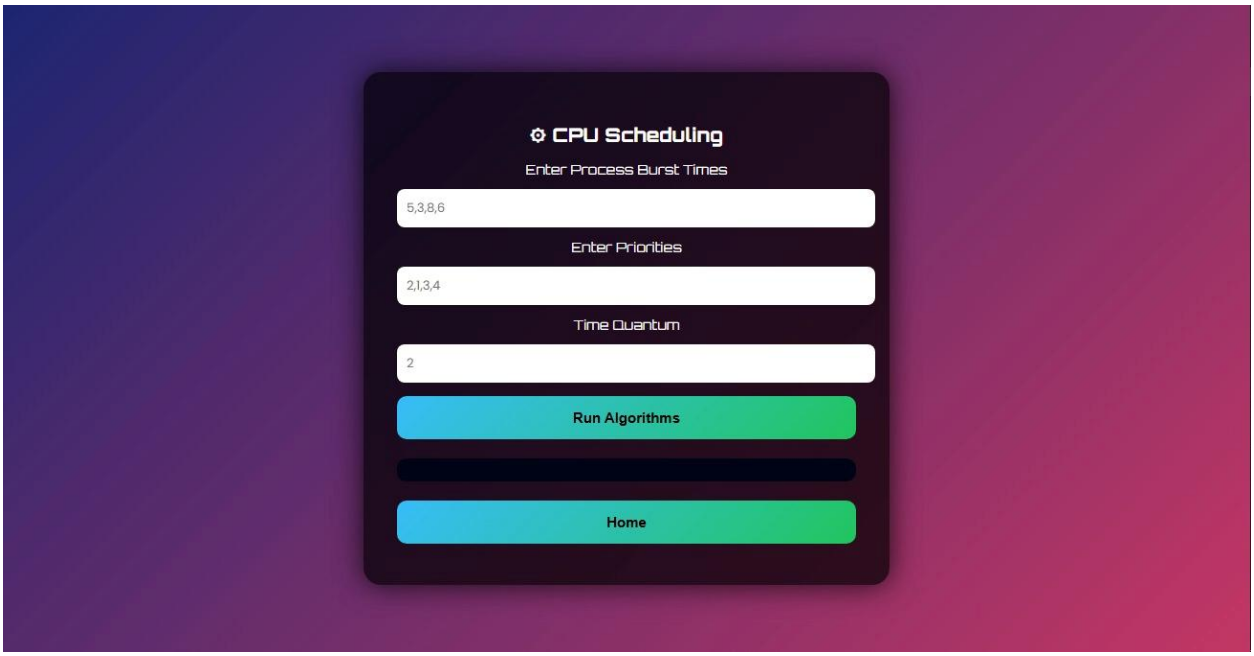
Deadlock module



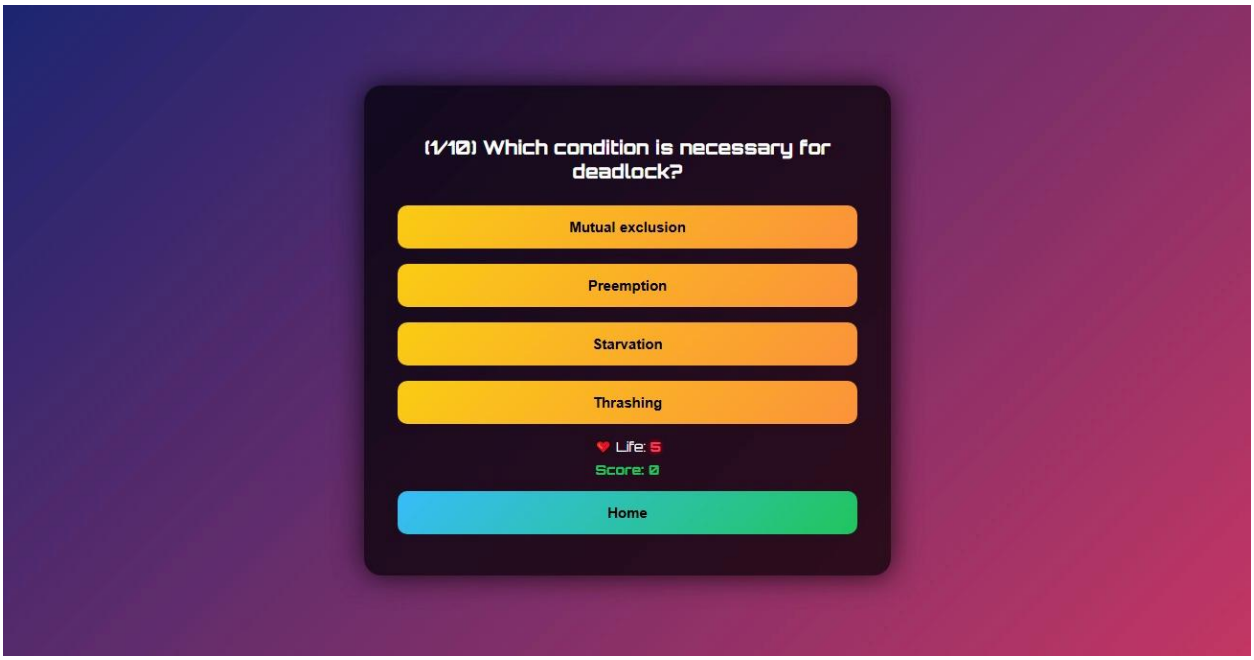
Memory allocation module



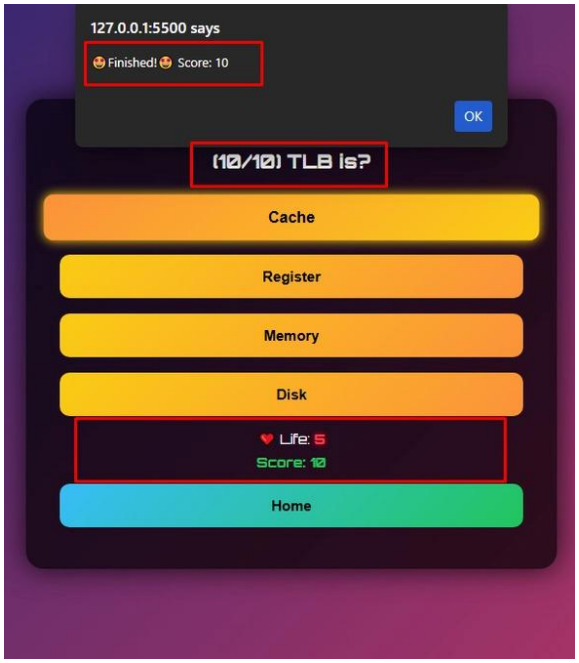
CPU scheduling module

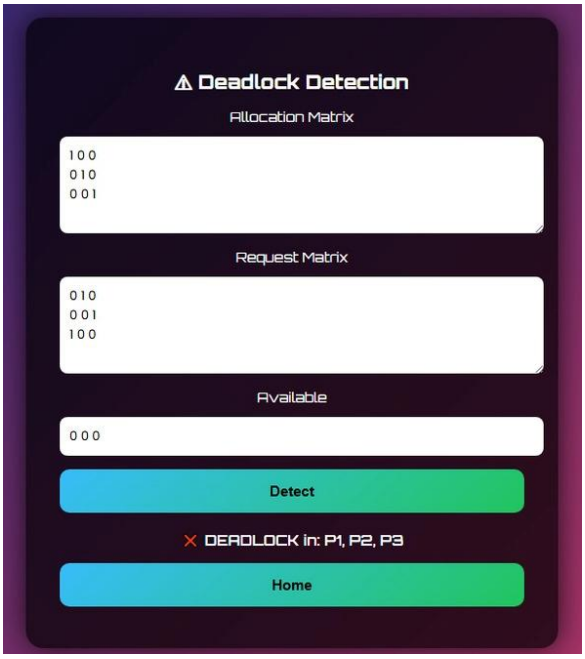


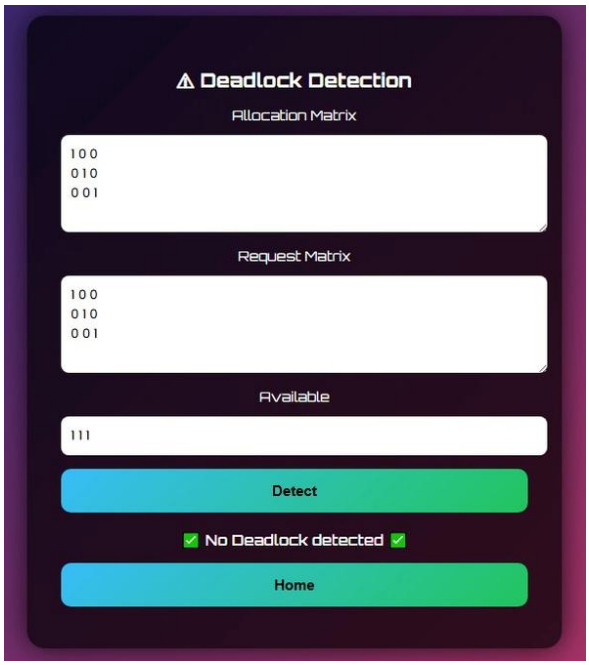
Quiz interface

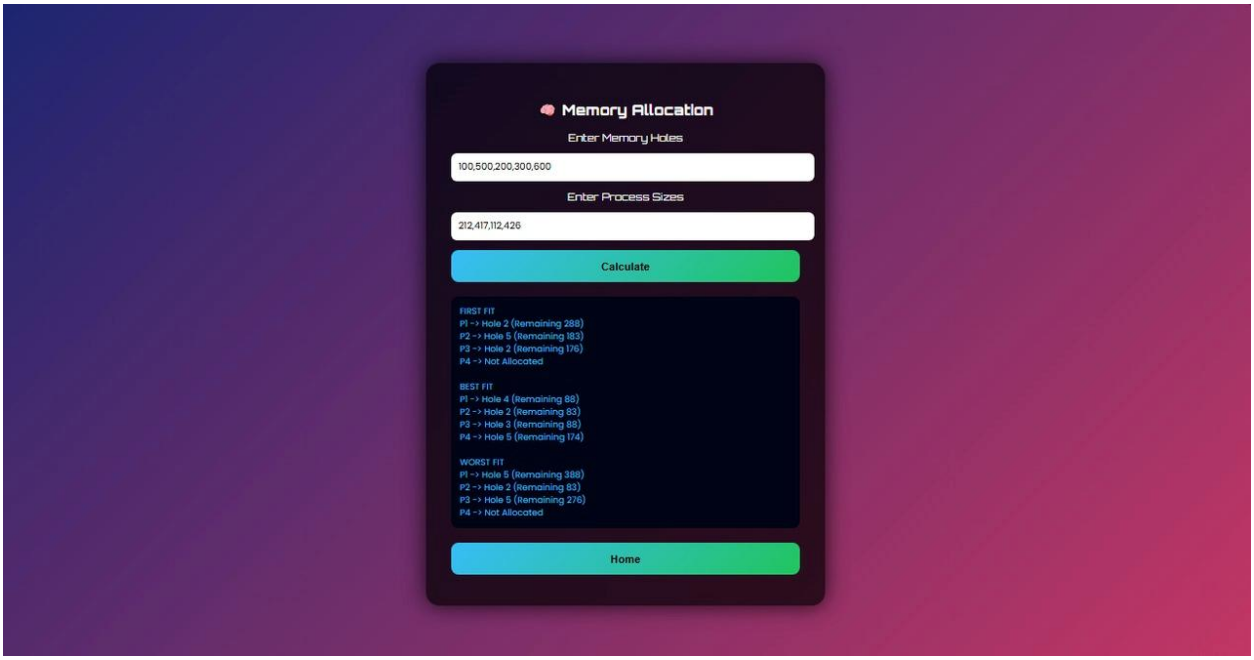


Output screens




⚠ Deadlock Detection
Allocation Matrix
1 0 0
0 1 0
0 0 1
Request Matrix
0 1 0
0 0 1
1 0 0
Available
0 0 0
Detect
✕ DEADLOCK in: P1, P2, P3
Home


⚠ Deadlock Detection
Allocation Matrix
1 0 0
0 1 0
0 0 1
Request Matrix
1 0 0
0 1 0
0 0 1
Available
1 1 1
Detect
✅ No Deadlock detected ✅
Home


🧠 Memory Allocation
Enter Memory Holes
100,500,200,300,600
Enter Process Sizes
212,417,112,426
Calculate
FIRST FIT
P1 -> Hole 2 (Remaining 288)
P2 -> Hole 5 (Remaining 183)
P3 -> Hole 2 (Remaining 176)
P4 -> Not Allocated
BEST FIT
P1 -> Hole 4 (Remaining 88)
P2 -> Hole 2 (Remaining 83)
P3 -> Hole 3 (Remaining 88)
P4 -> Hole 5 (Remaining 174)
WORST FIT
P1 -> Hole 5 (Remaining 388)
P2 -> Hole 2 (Remaining 83)
P3 -> Hole 5 (Remaining 276)
P4 -> Not Allocated
Home

10. Code Snape

index.html – UI Structure Snippet

```
<!-- Main Menu -->
<div id="home" class="box">
  <h2>OS Game System</h2>

  <button onclick="openQuiz()"> Quiz</button>
  <button onclick="openDeadlock()"> Deadlock</button>
  <button onclick="openMemory()"> Memory
Allocation</button>
  <button onclick="openCPU()">  CPU Scheduling</button>
</div>
```

```
<!-- Deadlock Section -->
<div id="deadlock" class="box hide">

<h3>Deadlock Detection</h3>

<textarea id="alloc" placeholder="Enter Allocation
Matrix"></textarea>
<textarea id="req" placeholder="Enter Request
Matrix"></textarea>
<input id="avail" placeholder="Available Resources">

<button onclick="detect()">Detect Deadlock</button>

<p id="result"></p>

</div>
```

```
<!-- Memory Allocation -->
<div id="memory" class="box hide">

<h3>Memory Allocation</h3>

<input id="holes" placeholder="100,500,200">
<input id="process" placeholder="212,417,112">

<button onclick="calculate()">Calculate</button>

<pre id="memResult"></pre>

</div>
```

```
<!-- CPU Scheduling -->
<div id="cpu" class="box hide">

<h3>CPU Scheduling</h3>

<input id="burst" placeholder="5,3,8,6">
<input id="priority" placeholder="2,1,3,4">
<input id="quantum" placeholder="2">

<button onclick="runCPU()">Run</button>

<pre id="cpuResult"></pre>

</div>
```

style.css – Design Snippet

```
body{
background:#0f172a;
font-family: Arial, sans-serif;
color:white;
}

.box{
background:#111827;
padding:20px;
border-radius:10px;
width:350px;
margin:auto;
text-align:center;
}

button{
background:#22c55e;
border:none;
padding:10px;
margin:5px;
color:black;
cursor:pointer;
border-radius:6px;
}

.hide{
display:none;
}
```

script.js – Core Logic Snippet

```
Menu Navigation
function openDeadlock(){
hideAll();
document.getElementById("deadlock").classList.remove("hide");
}

function openMemory(){
hideAll();
document.getElementById("memory").classList.remove("hide");
}

function openCPU(){
hideAll();
document.getElementById("cpu").classList.remove("hide");
}
```

```
function detect(){

let A = alloc.value.trim().split("\n")
.map(r => r.split(" ").map(Number));

let R = req.value.trim().split("\n")
.map(r => r.split(" ").map(Number));

let V = avail.value.split(" ").map(Number);

let n = A.length, m = V.length;
Deadlock Detection Logic

let finish = new Array(n).fill(false);
let work = [...V];

let progress = true;
```

```
while(progress){
progress = false;

for(let i=0;i<n;i++){
if(!finish[i]){

let can = true;

for(let j=0;j<m;j++){
if(R[i][j] > work[j]){
can = false; break;
}
}

if(can){
for(let j=0;j<m;j++)
work[j] += A[i][j];

finish[i] = true;
progress = true;
}
}
}
}

}
```

Memory Allocation

```
function fit(mem,proc,type){

for(let i=0;i<proc.length;i++){
let idx = -1;

for(let j=0;j<mem.length;j++){
if(mem[j] >= proc[i]){

if(type=="first"){idx=j;break;}

if(type=="best"){
if(idx==-1 || mem[j]<mem[idx]) idx=j;
}

if(type=="worst"){
if(idx==-1 || mem[j]>mem[idx]) idx=j;
}
}
}

if(idx!=-1)
mem[idx] -= proc[i];
}
}
```

CPU Scheduling Logic

```
function runCPU(){

let bt = burst.value.split(",").map(Number);
let pr = priority.value.split(",").map(Number);
let tq = Number(quantum.value);

fcfs(bt);
sjf(bt);
roundRobin(bt,tq);
priority(bt,pr);
}
```

11. Conclusion

The OS Game System provides an effective interactive platform for learning Operating System concepts. By integrating simulation, visualization, and assessment, the system enhances student engagement and understanding.
It is lightweight, browser-based, and user-friendly. This project serves as a valuable academic tool for OS education and laboratory practice.

11. Future Work

- ► Implement full Banker's Safety Algorithm
- ► Add paging and segmentation
- ► Introduce Gantt chart visualization
- ► Add waiting time & turnaround time
- ► Implement user login system
- ► Convert to mobile application

12. References

1. GeeksForGeeks – Operating System Tutorials
2. TutorialsPoint – OS Algorithms
3. MDN Web Docs – JavaScript Reference
4. Wikipedia – Operating System