南京信息工程大学　　实验（实习）报告

实验名称 最佳适应算法 日期 2023.11.30 指导教师 赵晓平

专业信息安全年级班级 21奇安信姓名朱宸扬学号 202183760012

一. 实验目的

深入理解最佳适应算法

二. 实验内容

模拟最佳适应算法

三. 实验原理

最佳适应算法

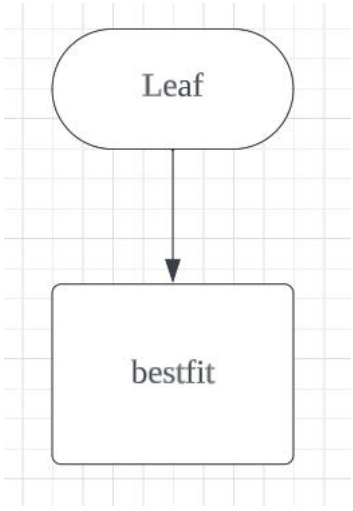
四. 实验设计及编码

1. 模块分析

进程，有自己的工作量

页面，最佳适应算法

2. 流程图



3. 代码实现

```
import pygame
import sys
```

```
class MemoryBlock:
    def __init__(self, size):
        self.size = size
        self.is_allocated = False
        self.process_id = None

def visualize_memory(memory_blocks):
    pygame.init()

    block_height = 50



    # screen_width = sum(block.size + margin for block in
memory_blocks) + margin
    screen_width = 820
    screen_height = block_height + 70



    screen = pygame.display.set_mode((screen_width + 200,
screen_height))
    pygame.display.set_caption('Memory Visualization')







    x_position = 0  # 将 x_position 移到循环外



    screen.fill((255, 255, 255))

    for block in memory_blocks:
        color = (100, 100, 100) if block.is_allocated else
(255,255,255)
```

```
        font = pygame.font.Font(None, 24)
        text_surface = font.render("White: Free, Grey: Allocated", True, (0, 0, 0))
        text_rect1 = text_surface.get_rect()
        text_rect1.center = (600, 30)


        # 将方块底部对准下方
        pygame.draw.rect(screen, color, (x_position, screen_height - block_height, block.size, block_height))

        font = pygame.font.Font(None, 36)
        text = font.render(str(block.size), True, (0, 0, 0))
        text_rect = text.get_rect(center=(x_position + block.size / 2, screen_height  - block_height / 2))
        screen.blit(text, text_rect)
        screen.blit(text_surface, text_rect1)
        #
        x_position += block.size



    pygame.display.flip()

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()

    pygame.time.wait(1000)
```

```
def best_fit(memory_blocks, process_size):
    best_fit_block = None

    for block in memory_blocks:
        if not block.is_allocated and block.size >= process_size:
            if best_fit_block is None or block.size <
best_fit_block.size:
                best_fit_block = block
    # 在 memory_blocks 里找一个最佳的空位，best_fit_block 为这个空位

    temp_size = best_fit_block.size
    if best_fit_block is not None:
        # 分配给进程
        best_fit_block.is_allocated = True
        best_fit_block.size = process_size
        best_fit_block.process_id = "Process"  # 模拟进程 ID

        # 创建新的空块
        remaining_size = temp_size - process_size
        if remaining_size > 0:
            remaining_block = MemoryBlock(remaining_size)
            index = memory_blocks.index(best_fit_block)
            memory_blocks.insert(index + 1, remaining_block)

        return best_fit_block
    else:
        return None


def deallocate(memory_blocks, process_id):
```

```
    for i, block in enumerate(memory_blocks):
        if block.is_allocated and block.process_id == process_id:
            # 释放进程占用的内存块
            block.is_allocated = False
            block.process_id = None

            # 合并相邻的空块
            if i < len(memory_blocks) - 1 and not memory_blocks[i +
1].is_allocated:
                block.size += memory_blocks[i + 1].size
                del memory_blocks[i + 1]

            if i > 0 and not memory_blocks[i - 1].is_allocated:
                block.size += memory_blocks[i - 1].size
                del memory_blocks[i - 1]
            break



def print_memory_status(memory_blocks):
    for i, block in enumerate(memory_blocks):
        status = "Allocated" if block.is_allocated else "Free"
        print(f"Block {i + 1}: Size = {block.size}, Status = {status},
Process ID = {block.process_id}")

def Pallocation(size,pool):

    allocated_block = best_fit(pool, size)

    if allocated_block is not None:
        print(f"Allocated {size} units to Block
{memory_blocks.index(allocated_block) + 1}")
    else:
        print(f"Unable to allocate {size} units")
    visualize_memory(pool)
```

```
    print("\n")

def PDeallocation(index, pool):
    if 0 <= index < len(pool):
        block_to_deallocate = pool[index]

        if block_to_deallocate.is_allocated:
            block_to_deallocate.is_allocated = False
            block_to_deallocate.process_id = None

            # 合并相邻的空块
            if index < len(pool) - 1 and not pool[index +
1].is_allocated:
                block_to_deallocate.size += pool[index + 1].size
                del pool[index + 1]

            if index > 0 and not pool[index - 1].is_allocated:
                pool[index - 1].size += block_to_deallocate.size
                del pool[index]

            print(f"Deallocated Block {index + 1}")
        else:
            print(f"Block {index + 1} is already Free")

        visualize_memory(pool)
    else:
        print(f"Invalid index: {index}")
if __name__ == "__main__":
    # 初始化内存块
    memory_blocks = [MemoryBlock(1000)]

    # 打印初始内存状态
    print("Initial Memory Status:")
    print_memory_status(memory_blocks)
```

```
print("\n")

Pallocation(50,memory_blocks)

Pallocation(200,memory_blocks)

Pallocation(300,memory_blocks)


PDeallocation(1,memory_blocks)

Pallocation(100,memory_blocks)
Pallocation(200,memory_blocks)
```

4. 结果及其相关分析（结果必须是图示）

另附

五． 实验小结

加深了对最佳适应算法的理解