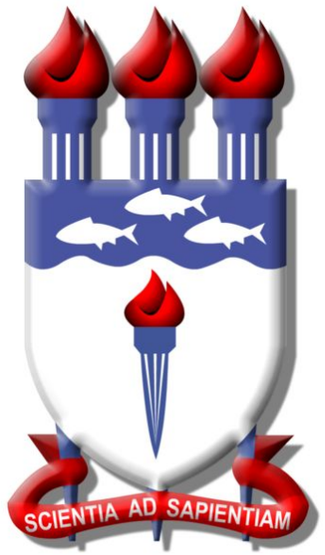
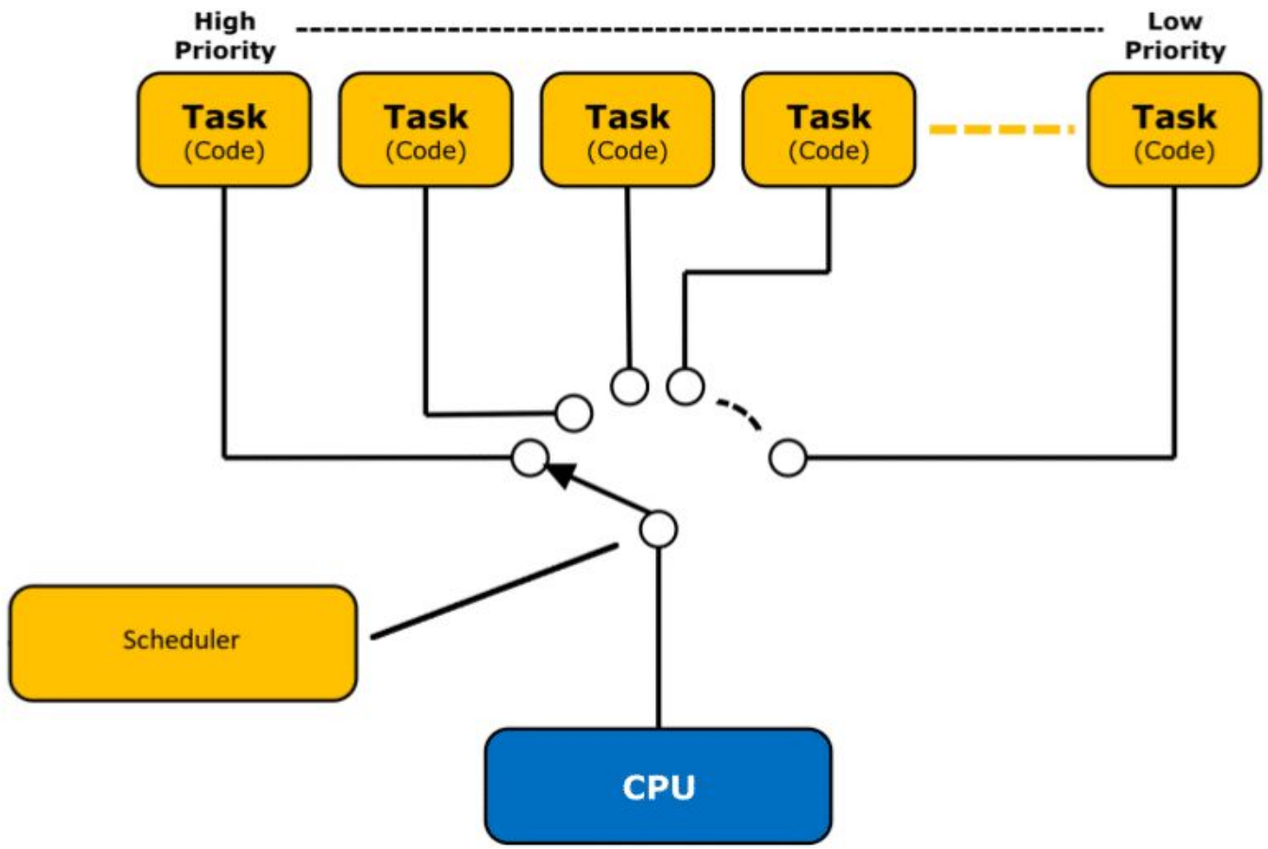


Agendamento de Processos

Andrey Noewertton - Ciência da Computação
Alexia Rodrigues - Ciência da Computação
Bárbara Albuquerque - Ciência da Computação
Maisa Barbosa - Engenharia da Computação
Ricardo Pinto - Engenharia da Computação

Problema

- Quem o processador deve executar primeiro?
  - Sistemas operacionais modernos precisam lidar com dezenas ou centenas de processos ativos simultaneamente.
  - Cada processo tem uma urgência ou importância diferente.
  - O sistema precisa escolher quem será executado primeiro — com rapidez e eficiência.



Desafios

- Processos interativos vs. processos em segundo plano.
- Manter tempo de resposta rápido para o usuário.
- Escalar para centenas de processos.
- Lidar com modificações nas prioridades.
- Garantir a execução de processos de baixa prioridade, evitando espera indefinida.

Código

```c
void aging(PriorityQueue *queue) {
    for (int i = 0; i < queue->size; i++) {
        queue->heap[i].wait_cycles++;

        if (queue->heap[i].type == 'i' && queue->heap[i].wait_cycles >= 2) {
            queue->heap[i].priority++;
            queue->heap[i].wait_cycles = 0;
        }
        else if (queue->heap[i].type == 'b' && queue->heap[i].wait_cycles >= 5) {
            queue->heap[i].priority++;
            queue->heap[i].wait_cycles = 0;
        }
    }
}
```

Código

```c
void execute_process(PriorityQueue *queue) {
    if (queue->size == 0) return;

    Process *p = &queue->heap[0];
    p->executed_cycles++;

    printf("\n Executando Processo ID #%d | Prioridade: %d | Ciclo %d/%d\n",
           p->id, p->priority, p->executed_cycles, p->required_cycles);

    if (p->required_cycles - p->executed_cycles <= 1) {
        if (p->type == 'i' && p->priority < 10) {
            printf("Processo Interativo ID #%d prestes a finalizar! Prioridade elevada para 10.\n", p->id);
            p->priority = 10;
        } else if (p->type == 'b' && p->priority < 9) {
            printf("Processo Background ID #%d prestes a finalizar! Prioridade elevada para 9.\n", p->id);
            p->priority = 9;
        }
    }

    if (p->executed_cycles >= p->required_cycles) {
        printf("Processo ID #%d concluido e removido da fila.\n", p->id);
        remove_max(queue);
    }
}
```

Código

```c
void generate_new_process(PriorityQueue *queue, int cycles) {
    if (cycles % 4 != 0) return;

    int new_id = rand() + cycles;
    int new_priority = rand() % 10 + 1;
    char type = (rand() % 2 == 0) ? 'i' : 'b';
    int required_cycles = (rand() % 5) + 1;

    printf("\n Novo processo chegou! (ID #%d, Prioridade: %d, Ciclos necessarios: %d)\n",
           new_id, new_priority, required_cycles);

    insert(queue, new_id, new_priority, type, required_cycles);
}
```

Código

```c
void insert(PriorityQueue *queue, int id, int priority, char type, int required_cycles) {
    int i = queue->size;

    queue->heap[i].id = id;
    queue->heap[i].priority = priority;
    queue->heap[i].wait_cycles = 0;
    queue->heap[i].executed_cycles = 0;
    queue->heap[i].required_cycles = required_cycles;
    queue->heap[i].type = type;

    while (i != 0) {
        int parent = (i - 1) / 2;

        if (queue->heap[parent].priority > queue->heap[i].priority) break;

        if (queue->heap[parent].priority == queue->heap[i].priority &&
            queue->heap[parent].type == 'i') break;

        swap(&queue->heap[i], &queue->heap[parent]);
        i = parent;
    }
    queue->size++;
}
```

Código

```c
void print_queue(PriorityQueue *queue) {
    printf("\n--- Estado Atual da Fila de Prioridade: ---\n");
    for (int i = 0; i < queue->size; i++) {
        printf("ID: %d | Prioridade: %d | Tipo: %s | Ciclos em espera: %d | Executado: %d/%d\n",
               queue->heap[i].id,
               queue->heap[i].priority,
               queue->heap[i].type == 'i' ? "Interativo" : "Background",
               queue->heap[i].wait_cycles,
               queue->heap[i].executed_cycles,
               queue->heap[i].required_cycles);
    }
    printf("-----------------------------------\n");
}
```

Código

```c
Process remove_max(PriorityQueue *queue) {
    Process max = queue->heap[0];
    queue->heap[0] = queue->heap[--queue->size];

    int i = 0;
    while (2 * i + 1 < queue->size) {
        int largest = i;
        int left = 2 * i + 1;
        int right = 2 * i + 2;

        if (queue->heap[left].priority > queue->heap[largest].priority ||
            (queue->heap[left].priority == queue->heap[largest].priority &&
             queue->heap[left].type == 'i' && queue->heap[largest].type != 'i')) {
            largest = left;
        }

        if (right < queue->size &&
            (queue->heap[right].priority > queue->heap[largest].priority ||
             (queue->heap[right].priority == queue->heap[largest].priority &&
              queue->heap[right].type == 'i' && queue->heap[largest].type != 'i'))) {
            largest = right;
        }

        if (largest == i) break;

        swap(&queue->heap[i], &queue->heap[largest]);
        i = largest;
    }
    return max;
}
```

Código

```c
int main() {
    PriorityQueue queue = create_priority_queue();
    int cycles = 0;

    insert(&queue, 1, 3, 'i', 4);
    insert(&queue, 2, 2, 'b', 5);
    insert(&queue, 3, 4, 'i', 2);

    print_queue(&queue);

    printf("\nIniciando Agendamento de Processos...\n");

    while (queue.size > 0) {
        printf("\n--- Ciclo numero: %d ---\n", cycles);

        generate_new_process(&queue, cycles);
        execute_process(&queue);
        cycles++;
        aging(&queue);
        print_queue(&queue);
    }

    printf("\nTodos os processos foram concluidos.\n");
    return 0;
}
```

De volta à Motivação…

- O sistema operacional lida com diversos processos de diferentes níveis de urgência e importância.

- Esses processos devem ser classificados em diferentes níveis de prioridade.

- A utilização de uma fila de prioridade implementada como uma heap permite classificar e gerenciar esses processos, de modo que o sistema possa selecionar e executar, primeiramente, aqueles com maior necessidade imediata para o sistema ou para o usuário.

Referências

- Normando, C. (2024, março 6). Gerenciamento de processos. Medium. https://medium.com/@celionormando/gerenciamento-de-processos-b7ccc3737bb0
- Priority Queue using Binary Heap. (2020, setembro 16). GeeksforGeeks. https://www.geeksforgeeks.org/priority-queue-using-binary-heap/
- No title. ([s.d.]). Cplusplus.com. https://cplusplus.com/reference/queue/priority_queue/
- Alves, J. (2024, fevereiro 21). Sistemas Operacionais – Gerenciamento de processos. Gran Cursos Online. https://blog.grancursosonline.com.br/sistemas-operacionais-gerenciamento-de-processos/
- CPU Scheduling in Operating Systems. (2025, abril 04). GeeksforGeeks. https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/