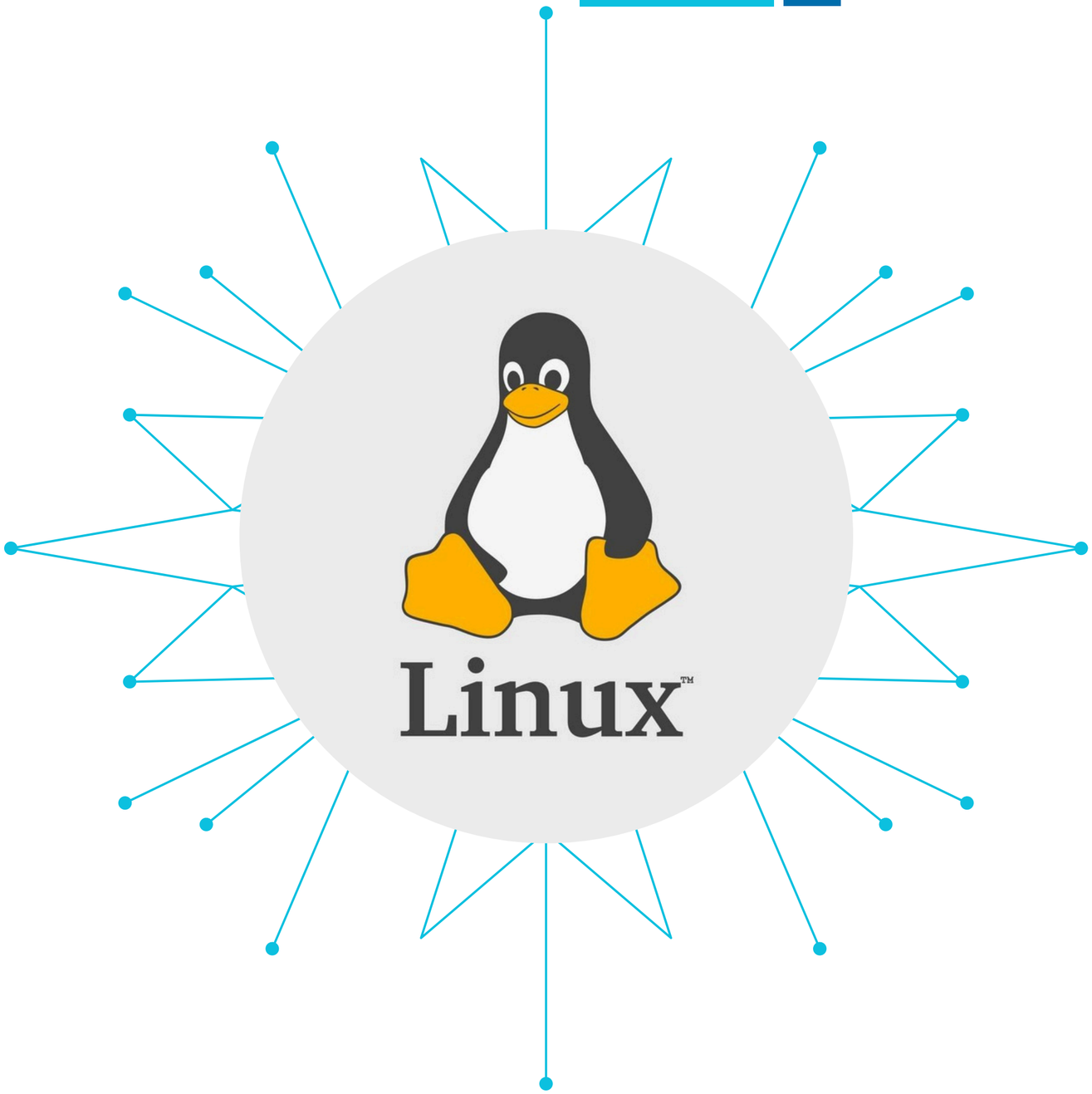
SCHEDULING POLICY
LINUX KERNEL

José Victor Rocha de Alencar
Luiz Gustavo Dall'Agnol Cavalcante

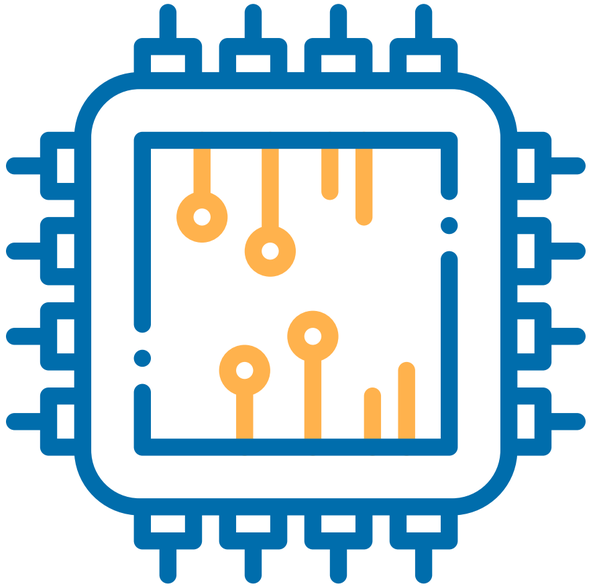


OBJETIVOS
DO TRABALHO

- Definir e explicar o escalonador do Linux
- Apresentar políticas de escalonamento
- Mostrar como criar uma nova política de escalonamento
- Implementar e avaliar uma nova política

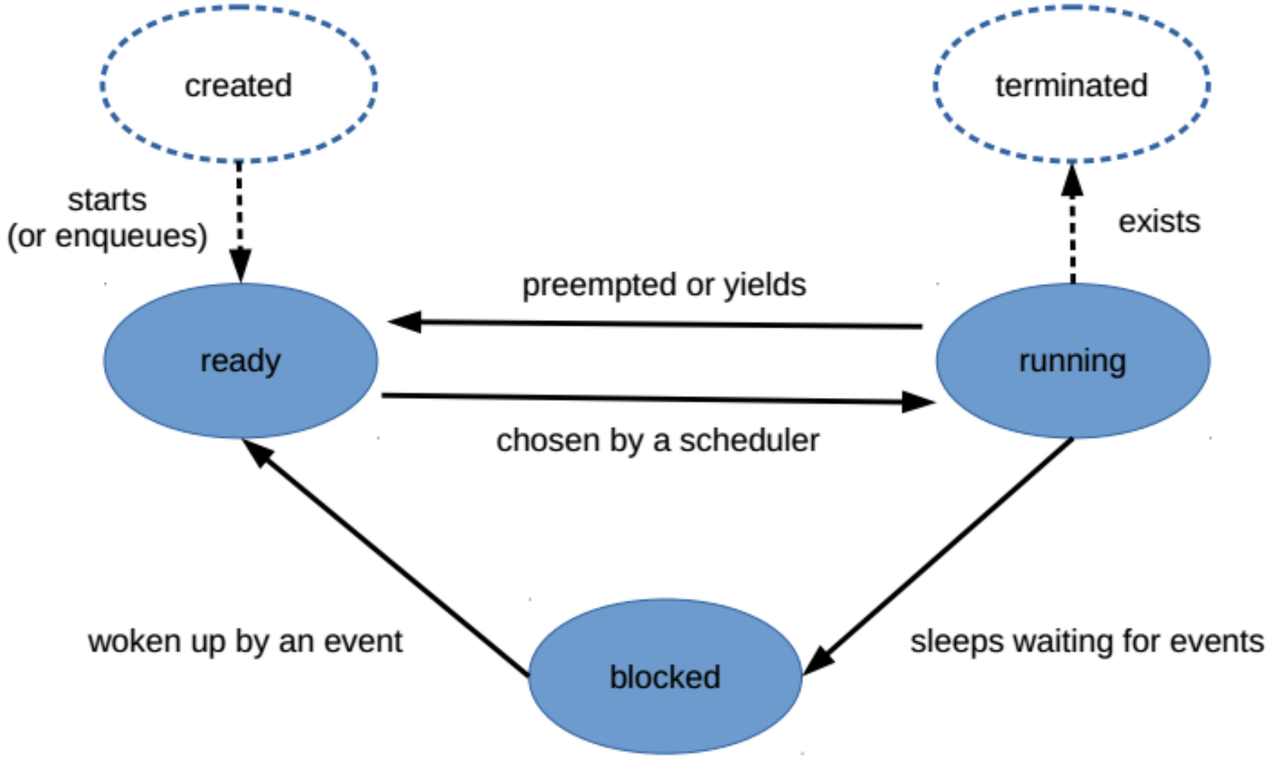
O QUE É O **ESCALONADOR?**

- Componente responsável por definir quais processos serão executados pela CPU e quando.
- Distribuição vária de acordo com a política.
- Política varia de acordo com a função.
- Modelo preemptivo
- Classifica as Threads

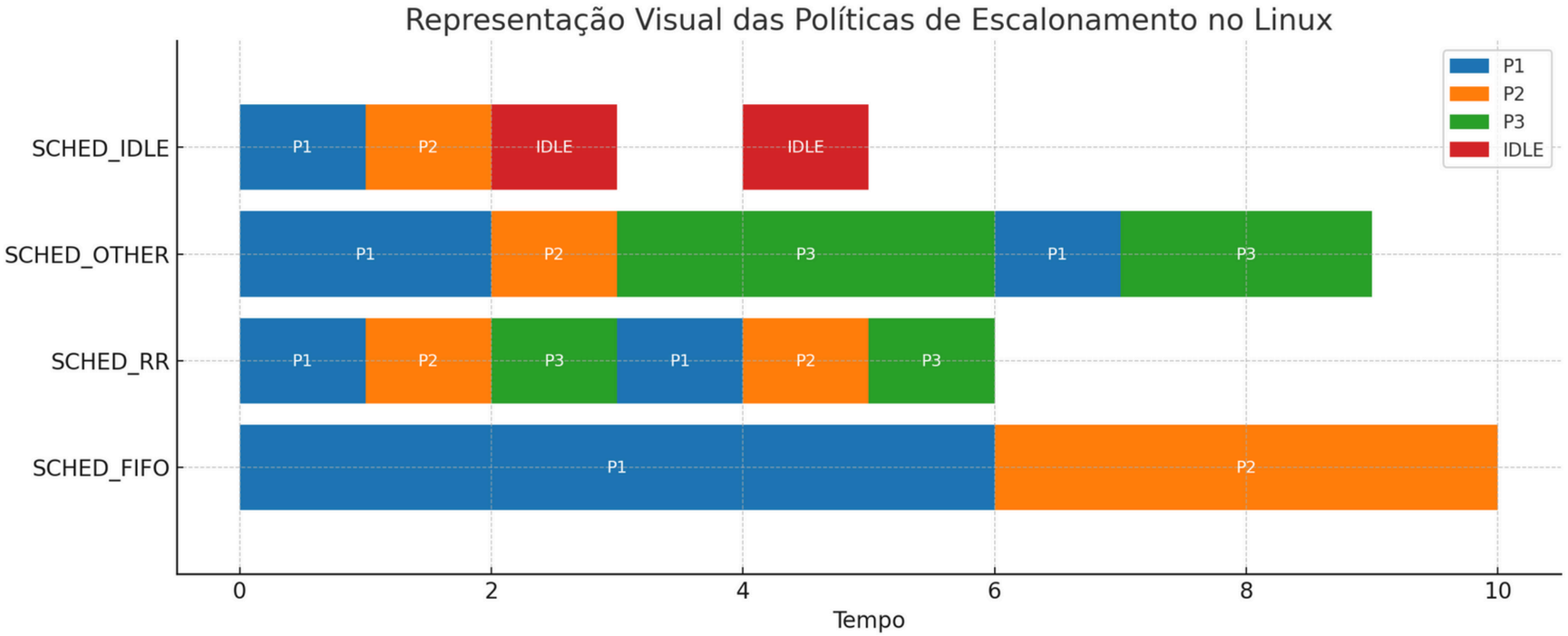


CLASSIFICAÇÃO DAS THREADS

- Estado: Ready, Running ou Blocked
- Utilização de Recursos: I/O-Bound e CPU-Bound
- Tempo: Real ou Não Real
- Entre outras características



POLÍTICAS DE ESCALONAMENTO



TUTORIAL

- **Ubuntu no WSL2 e QEMU**
  - Instale o wsl2 (distro pode ser ubuntu ou debian) e o qemu. Atualize usando o ‘sudo apt update’.
- **Baixe o código fonte do kernel com wget**
  - **No terminal “** wget https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/snapshot/linux-5.15.tar.gz**”**
- **Defina a política**
  - #define SCHED_BACKGROUND 7
- **Registre a nova classe de escalonamento em kernel/sched/core.c:**
  - extern const struct sched_class background_sched_class;
- **Localize as linhas:**
  - static const struct sched_class *sched_class_highest = &stop_sched_class;
  - static const struct sched_class *sched_class_lowest = &idle_sched_class;
- **Modifique para:**
  - static const struct sched_class *sched_class_highest = &stop_sched_class;
  - static const struct sched_class *sched_class_lowest = &background_sched_class;
- **Implemente a classe nova**
  - **#include “sched.h”**
  - **const struct sched_class background_sched_class ={//reaproveita o algoritmo CFS}**

TUTORIAL

- **Atualize o mapeamento de política**
  - Ainda em kernel/sched/core.c na linha sched_class:
    - case SCHED_BACKGROUND:
      - return &background_sched_class;
- **Valide**
  - #define sched_policy_valid(policy) \
  - ((policy) == SCHED_NORMAL || \
  - *demais políticas
- **Compile**
  - make defconfig
  - make -j$(nproc)
- **Execute no QEMU**
  - qemu-system-x86_64 \
  - -kernel arch/x86/boot/bzImage \
  - -append "console=ttyS0" \
  - -nographic \
  - -m 1G
- **Teste a política**
  - dentro de #define _GNU_SOURCE
  - gcc test_bg.c -o test_bg

IMPLEMENTAÇÃO

NOVA POLÍTICA

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sched.h>
#include <errno.h>
#include <string.h>

int main(int argc, char *argv[]) {
    if (argc < 3) {
        fprintf(stderr, "Uso: %s <politica> <caminho_programa>\n", argv[0]);
        return 1;
    }

    int policy;
    if (strcmp(argv[1], "OTHER") == 0) policy = SCHED_OTHER;
    else if (strcmp(argv[1], "FIFO") == 0) policy = SCHED_FIFO;
    else if (strcmp(argv[1], "RR") == 0) policy = SCHED_RR;
    else if (strcmp(argv[1], "BACKGROUND") == 0) policy = 7;
    else {
        fprintf(stderr, "Política desconhecida.\n");
        return 1;
    }

    struct sched_param param;
    param.sched_priority = (policy == SCHED_OTHER || policy == 7) ? 0 : 10;

    if (sched_setscheduler(0, policy, &param) < 0) {
        perror("sched_setscheduler");
        return 1;
    }

    execv(argv[2], &argv[2]);
    perror("execv");
    return 1;
}
```

BENCHMARK

```c
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sched.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <errno.h>

void run_cpu_intensive_task() {
    volatile unsigned long long i, sum = 0;
    for (i = 0; i < 1000000000ULL; i++) {
        sum += i;
    }
}

void print_times(struct timeval start, struct timeval end, struct rusage usage) {
    long wallclock = (end.tv_sec - start.tv_sec) * 1000 + (end.tv_usec - start.tv_usec) / 1000;
    long user = usage.ru_utime.tv_sec * 1000 + usage.ru_utime.tv_usec / 1000;
    long sys  = usage.ru_stime.tv_sec * 1000 + usage.ru_stime.tv_usec / 1000;

    printf("Wallclock Time: %ld ms\n", wallclock);
    printf("User Time: %ld ms\n", user);
    printf("System Time: %ld ms\n", sys);
}

int main() {
    struct timeval start, end;
    struct rusage usage;

    gettimeofday(&start, NULL);
    run_cpu_intensive_task();
    gettimeofday(&end, NULL);
    getrusage(RUSAGE_SELF, &usage);

    print_times(start, end, usage);
    return 0;
}
```

CONCLUSÃO

- Operação em segundo-plano
- Pequeno impacto no desempenho
- Perfomance semelhante ao SCHED_OTHER
- Ausência de política de envelhecimento

OBRIGADO
PELA ATENÇÃO