Dokumentacja projektu

**Resource Monitor**

Mateusz Bocak

Systemy Operacyjne 2

III rok informatyki stacjonarnie

# 1. Opis działania

Resource Monitor to narzędzie służące do monitorowania w czasie rzeczywistym zużycia zasobów systemowych przez procesy w systemie Linux. Program śledzi wykorzystanie CPU oraz pamięci RAM dla wskazanego procesu, wyświetlając aktualizowane co sekundę statystyki.

## Główne funkcjonalności

* Monitorowanie zużycia CPU w procentach
* Śledzenie wykorzystania pamięci RAM w MB
* Automatyczne wykrywanie zakończenia monitorowanego procesu
* Obsługa różnych typów zakończenia procesu (normalne, przez sygnał)

## Sposób użycia

*./resource-monitor "polecenie [argumenty...]"*

# 2. Plan aplikacji i analiza implementacji

## 2.1 Kluczowe komponenty

### Struktura ResourceUsage

typedef struct {

    double cpu\_usage;

    long memory\_usage;

} ResourceUsage;

Struktura przechowująca podstawowe metryki monitorowanego procesu.

### Główne funkcje

1. *parse\_command()* - parsowanie argumentów wiersza poleceń
2. *get\_cpu\_usage()* - obliczanie wykorzystania CPU
3. *get\_memory\_usage()* - pomiar zużycia pamięci
4. *monitor\_process()* - główna pętla monitorowania
5. *main()* - inicjalizacja i zarządzanie cyklem życia procesu

## 2.2 Trudności implementacyjne

### Pomiar wykorzystania CPU

Największym wyzwaniem było poprawne obliczanie procentowego wykorzystania CPU. Wymagało to:

* Dostępu do statystyk procesu w */proc/[pid]/stat*
* Przetwarzania wartości *utime* i *stime*
* Uwzględnienia częstotliwości taktowania CPU *(sysconf(\_SC\_CLK\_TCK))*
* Obliczania różnicy między kolejnymi pomiarami

Implementacja w kodzie:

double total\_time = (utime + stime) / (double)sysconf(\_SC\_CLK\_TCK);

…

return total\_time \* 100.0;

### Monitorowanie pamięci

Wyzwania przy implementacji monitorowania pamięci:

* Parsowanie pliku */proc/[pid]/status*
* Prawidłowa interpretacja pola *VmRSS*
* Konwersja jednostek (KB na MB)

### Zarządzanie procesami

Kluczowe aspekty:

* Wykorzystanie *fork()* do tworzenia procesu potomnego
* Prawidłowa obsługa *execvp()* dla uruchamiania poleceń
* Monitorowanie stanu procesu za pomocą *kill(pid, 0)*
* Obsługa różnych scenariuszy zakończenia procesu

# 3. Opis testów

## 3.1 Zakres testów

Test suite obejmuje 8 różnych scenariuszy:

1. Długotrwały proces (sleep 5)
2. Krótkotrwały proces (sleep 1)
3. Nieistniejące polecenie
4. Proces kończący się przed rozpoczęciem monitorowania
5. Oczekiwanie na zakończenie procesu
6. Obciążenie CPU (openssl speed)
7. Obciążenie I/O (cat /dev/zero)
8. Intensywne obliczenia (sha1sum)

## 3.2 Mechanizm testowania

* Wykorzystanie skryptu bash do automatyzacji
* Weryfikacja kodów powrotu
* Obsługa procesów w tle
* Czyszczenie procesów testowych

## 3.3 Funkcje pomocnicze testów

* *cleanup()* - zatrzymywanie procesów w tle
* *report\_test()* - raportowanie wyników

## 3.4 Raport testów

# 4. Research techniczny

## 4.1 System budowania i kompilacja

### Makefile

Wykorzystano standardowy system budowania GNU Make z następującą konfiguracją:

* Organizacja katalogów: *src/, obj/, bin/*
* Flagi kompilatora: *-Wall -Wextra -O2* dla debugowania i optymalizacji
* Automatyczne wykrywanie zależności
* Target *'clean'* do czyszczenia plików tymczasowych

Implementacja bazuje na dokumentacji GNU Make [1]: <https://www.gnu.org/software/make/manual/>

### Kompilator i flagi

Wykorzystano GCC z następującymi flagami:

* *-Wall* i *-Wextra*: wykrywanie potencjalnych błędów
* *-O2*: optymalizacja kodu
* *-D\_GNU\_SOURCE*: dostęp do rozszerzeń GNU/Linux

Szczegóły flag kompilatora: GCC manual [2]: <https://gcc.gnu.org/onlinedocs/>

## 4.2 Mechanizmy systemowe

### System plików /proc

Wykorzystano wirtualny system plików /proc do monitorowania procesów:

#### /proc/[pid]/stat

double get\_cpu\_usage(pid\_t *pid*) {

    char stat\_path[64];

    snprintf(stat\_path, sizeof(stat\_path), "/proc/%d/stat", *pid*);

    // …

}

* Źródło dokumentacji: Linux Kernel Documentation [3]: <https://www.kernel.org/doc/Documentation/filesystems/proc.txt>

#### /proc/[pid]/status

long get\_memory\_usage(pid\_t *pid*) {

    char status\_path[64];

    snprintf(status\_path, sizeof(status\_path), "/proc/%d/status", *pid*);

// …

}

* Format pliku opisany w: proc(5) man page [4]

### Funkcje POSIX

#### Zarządzanie procesami

pid\_t pid = fork(); // Tworzenie procesu potomnego

if (pid == 0) {

    execvp(cmd\_args[0], cmd\_args); // Wykonanie nowego procesu

    perror("Exec failed");

    cleanup\_args(cmd\_args);

    return 1;

}  
  
…  
  
int status;

waitpid(pid, &status, 0); // Oczekiwanie na zakończenie procesu potomnego

* Implementacja bazuje na: POSIX.1-2017 standard [5]

#### Obsługa sygnałów

if (kill(*pid*, 0) == -1) {

    if (errno == ESRCH) {

        printf("Process has terminated.\n");

        break;

    } else {

        perror("Error checking process status");

        break;

    }

}

* Dokumentacja: signal(7) man page [6]

## Funkcje pomocnicze

### Zarządzanie pamięcią

#### Dynamiczna alokacja

char\*\* args = malloc(sizeof(char\*) \* MAX\_CMDLINE);

// …

free(*args*[i]); // w osobnej funkcji do czyszczenia argumentów

#### Operacje na stringach

char\* cmd\_copy = strdup(*cmd*); *// Bezpieczne kopiowanie*

char\* token = strtok(cmd\_copy, " "); *// Parsowanie argumentów*

* Bazuje na: C Standard Library documentation [7]

## Obsługa błędów

Zaimplementowano wielopoziomową obsługę błędów.

#### Błędy systemowe

if (pid < 0) {

    perror("Fork failed");

    cleanup\_args(cmd\_args);

    return 1;

}

if (pid == 0) {

    execvp(cmd\_args[0], cmd\_args);

    perror("Exec failed");

    cleanup\_args(cmd\_args);

    return 1;

}

#### Analiza statusu zakończenia

if (WIFEXITED(status)) {

    return WEXITSTATUS(status);

} else if (WIFSIGNALED(status)) {

    fprintf(stderr, "Process terminated by signal %d\n", WTERMSIG(status));

    return 1;

} else {

    fprintf(stderr, "Process terminated abnormally\n");

    return 1;

}

* Szczegóły w: wait(2) man page [8]

# Bibliografia

* [1] GNU Make Manual: <https://www.gnu.org/software/make/manual/>
* [2] GCC Documentation: <https://gcc.gnu.org/onlinedocs/>
* [3] Linux Kernel /proc Documentation: <https://www.kernel.org/doc/Documentation/filesystems/proc.txt>
* [4] proc(5) Manual Page: <http://man7.org/linux/man-pages/man5/proc.5.html>
* [5] POSIX.1-2017 standard: <https://pubs.opengroup.org/onlinepubs/9699919799/>
* [6] signal(7) Manual Page: <http://man7.org/linux/man-pages/man7/signal.7.html>
* [7] C Standard Library: <https://en.cppreference.com/w/c/memory>
* [8] wait(2) Manual Page: <http://man7.org/linux/man-pages/man2/wait.2.html>