## PROPOSAL

**ANALISIS PERBANDINGAN ALGORITMA PENJADWALAN *ROUND***

### ROBIN DAN SHORTEST JOB FISRT UNTUK MANAJEMEN PROSES DALAM SINGLE PROCESSING

Diajukan untuk Memenuhi Salah Satu Syarat

Memperoleh Gelar Sarjana Teknik (S.T.) Program Studi Teknik Informatika



OLEH :

**LA ODE MUHAMAD TAUFIQ**

**E1E1 15 025**

**JURUSAN TEKNIK INFORMATIKA**

**FAKULTAS TEKNIK UNIVERSITAS HALU OLEO**

**KENDARI**

**2020**

**BAB 1**

## PENDAHULUAN

**1.1 Latar Belakang**

Komputer bisa membantu manusia menyelesaikan sebuah perintah pada tingkat yang rumit sekalipun. Untuk bisa bekerja dengan baik, komputer membutukan banyak perangkat di dalamnya. Komputer memerlukan perangkat lunak dan perangkat keras yang berkolaborasi dengan tepat untuk bisa berjalan. Sistem operasi atau biasa disingkat dengan OS *(Operation System)* adalah salah satu perangkat yang harus ada pada komputer. Tanpa sistem operasi, komputer tidak akan dapat bekerja. Adanya sistem operasi membantu komputer untuk mengatur seluruh sumber daya yang ada pada komputer. Sistem operasi merupakan nyawa dari komputer, jika tidak ada pengoperasian, maka komputer tidak dapat berjalan lancar.

Seiring dengan perkembangan ilmu pengetahuan teknologi, informasi, dan komunikasi, tuntutan dalam mengelola suatu proses dalam sistem operasi adalah suatu hal yang sangat di butuhkan pada zaman ini. Dalam kegiatannya sistem operasi memilki mekanisme ptoteksi untuk memastikan dirinya, program yang berjalan, dan data-data penggunaannya berjalan dengan baik. Sistem operasi melakukan beragam tugas, dan salah satu yang paling penting adalah manajemen proses, dimana mengelola semua proses aktif dan mengalokasikan sumber daya ke proses-proses itu sesuai kebijaksanaan yang di ambil untuk memenuhi sasaran kerja CPU.

Manajemen proses merupakan konsep pokok dalam sistem operasi, sehingga masalah manajemen proses adalah masalah utama dalam perancangan manajemen proses. Proses adalah sebuah program yang sedang di eksekusi yang mencakup *program counter*, *register*, dan variabel di dalamnya (Tanenbaum dan Woodhull 2001) . Sedang program adalah kumpulan intruksi yang di tulis kedalam bahasa yang dimengerti sistem operasi. Sebuah proses membutuhkan sejumlah sumber daya untuk menyelesaikan tugasnya, berupa CPU *time*, alamat memori, dan berkas-berkas.

Sistem operasi mengalokasikan sumber-sumber daya tersebut saat sistem operasi dijalankan atau diproses. Sistem operasi bertanggung jawab atas aktivitasaktivitas yang berkaitan dengan manajemen proses seperti, membuat dan menghapus proses pengguna dan sistem proses, menunda atau melanjutkan proses, menyediakan mekanisme untuk sinkronisasi, menyediakan mekanisme untuk komunikasi, dan menyediakan mekanisme untuk penanganan.

Penjadwalan proses merupakan kumpulan kebijaksanaan dan mekanisme sistem operasi yang berkaitan dengan urutan kerja yang dilakukan sistem komputer (Tanenbaum dan Woodhull 2001). Untuk memutuskan proses yang harus berjalan, kapan dan seberapa lama proses itu berjalan di perlukan suatu algoritma penjadwalan proses yang efektif. Penjadwalan proses yang efektif, memiliki kriteria untuk mengukur dan optimasi kinerja diantaranya adil (*fairnes*), efisiensi (*eficiency*), waktu tanggap (*response time*), *turnaround time*, dan *throughput* (Tarek 2006). Persoalan penjadwalan proses dapat di selesaikan dengan beberapa algoritma penjadwalan, dimana setiap algoritma mempunyai kriteria dan keunggulannya masing-masing. Ada berbagai macam algoritma penjadwalan proses di dalam CPU dan dua diantaranya adalah algoritma *round robin* dan algoritma *shortest job first* yang akan di bahas pada tugas akhir ini.

Algoritma *round robin* merupakan algoritma yang bergantung pada *quantum time*. Jika *quantum time* terlalu besar, maka *response time* untuk proses yang lain terlalu tinggi. Sebaliknya jika *quantum time* terlalu kecil maka akan mengakibatkan *overhead* pada CPU dimana *context switching* lebih besar. Algoritma *shortest job first* adalah algoritma penjadwalan yang optimal dengan rata-rata waktu tunggu yang minimal. Namun pada kenyataannya sulit di implementasikan karena sulit untuk mengetahui panjang CPU *burst* berikutnya. Sampai saat ini *user* tidak mengetahui berapa lama sebuah job atau proses dikerjakan pada algoritma *round robin* dan *shortest job first* dalam sebuah komputer untuk manajemen proses dalam *single processing*.

Berdasarkan penjelasan latar belakang diatas penulis mengangkat judul ***“*Analisis Perbandingan Algoritma Penjadwalan *Round Robin* dan *Shortest Job First* untuk Manajemen Proses dalam *Single Processing*”**.

**1.2 Rumusan Masalah**

Berdasarkan latar belakang yang telah di uraikan di atas, dapat di rumuskan permasalahan dalam penelitian ini yaitu :

1. Bagaimana perancangan aplikasi simulasi algoritma penjadwalan *round robin* dan *shortest job first* untuk manajemen proses dalam *single processing*.
2. Bagaimana mengukur kinerja algoritma *round robin* dan *shortest job first* untuk manajemen proses dalam s*ingle processing.*
3. Bagaimana tahap analisis perbandingan algoritma *round robin* dan *shortest job first* untuk manajemen proses dalam *single processing* akan dilakukan.

**1.3 Batasan Masalah**

Adapun batasan masalah dalam penelitian ini yaitu :

1. Analisis perbandingan algoritma penjadwalan *round robin* dan *shortest job first* berdasarkan pada rata-rata *turnaround time* dan *waiting time* (dalam bentuk grafik) serta perbandingan secara visual terhadap *gant chart* dari dua algoritma tersebut.
2. Bahasa pemrograman yang di gunakan pada perancangan aplikasi simulasi adalah bahasa pemrograman JAVA dengan menggunakan aplikasi *Netbeants* IDE 8.2.
3. Pada aplikasi simulasi, *user* dapat menambah proses baru, memodifikasinya serta menghapusnya, dan semua data simulasi akan diperbaharui secara otomatis.
4. Pada aplikasi simulasi, algoritma *round robin* bisa memiliki lebih dari satu simulasi dengan nilai *quantum time* yang berbeda. *User* dapat menentukan sendiri daftar *quantum time* yang diinginkan dan secara otomatis aplikasi akan menampilkan simulasi *round robin* untuk setiap *quantum time* yang telah ditentukan tersebut.
5. Pada aplikasi simulasi, *user* dapat melihat animasi yang memvisualisasikan kegiatan penjadwalan proses. *User* juga dapat mengontrol jalannya animasi yang ditampilkan. Animasi akan diperbaharui secara otomatis apabila terjadi perubahan konfigurasi.
6. Pada aplikasi simulasi, pengguna dapat mengetahui status dari setiap proses pada setiap CPU *time* yang ditampilkan dalan bentuk tabel.
7. Parameter dalam pengujian yaitu dengan melihat nilai *average waiting time* dan *average turnaround time* dari enam paremeter yang ada, dalam mengetahui kompleksitas pada algoritma penjadwalan.

**1.4 Tujuan Penelitian**

Adapun tujuan dalam penelitian ini yaitu :

1. Untuk merancang aplikasi simulasi penjadwalan *round robin* dan *shortest job first* untuk manajemen proses dalam *single processing*.
2. Untuk mengukur kinerja algoritma penjadwalan *round robin* dan *shortest job first* untuk manajemen proses dalam *single processing*.
3. Untuk menganalisis perbandingan algoritma *round robin* dan *shortest job first* untuk manajemen proses dalam *single processing.*

* 1. **Manfaat Penelitian**

Manfaat penelitian ini yaitu membantu *user* dalam menentukan algoritma penjadwalan yang lebih baik dan efektif dan memberikan pemahaman pada *user* tentang konsep algoritma penjadwalan yang disimulasikan .

* 1. **Sistematika Penelitian**

Penyusunan tugas akhir ini menggunakan sistematika penulisan :

**BAB I PENDAHULUAN**

Bab ini berisi latar belakang, rumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, sistematika penelitian dan tinjauan pustaka.

**BAB II LANDASAN TEORI**

Bab ini berisi sejarah sistem operasi, pengertian sistem operasi, defenisi proses, defenisi manajemen proses, defenisi *single processing,* pengertian algoritma *round robin*, konsep dasar algoritma *round robin*, pengertian algoritma *shortest job first* dan konsep dasar algoritma *shortest job first*.

**BAB III METODOLOGI PENELITIAN**

Bab ini berisi prosedur dan pengumpulan data, prosedur pengembangan sistem serta waktu dan tempat penilitian.

**BAB IV ANALISIS DAN PERANCANGAN SISTEM**

Bab ini berisi penjelasan tentang perancangan serta analisis perbandingan algoritma penjadwalan dalam sistem.

**BAB V IMPLEMENTASI DAN PENGUJIAN**

Bab ini memuat aplikasi simulasi dari hasil penelitian, analisis perbandingan algoritma, dan pembahasan masalah yang dihadapi.

**BAB IV PENUTUP**

Bab ini menguraikan kesimpulan dari keseluruhan penelitian serta memberikan saran-saran yang berguna untuk mengatasi masalah yang dihadapi.

**1.7 Tinjauan Pustaka**

Penelitian ini dibuat berdasarkan penelitian yang pernah dilakukan oleh

Masrizal (2017) yang berjudul “Simulasi Perbandingan Penjadwalan *Round Robin* dan FCFS untuk Manajemen Proses dalam *Single Processing*”, yang membahas tentang perbandingan algoritma *round robin* dengan FCFS menggunakan aplikasi simulasi. Hasil penelitian menunjukan bahwa algoritma penjadwalan FCFS lebih efesien dibandingkan algoritma penjadwalan *round robin* karena algoritma penjadwalan *round robin* menggunakan *quantum* dalam proses antrian.

Penelitian ini dibuat berdasarkan penelitian yang pernah dilakukan oleh

Sonri et al. (2013) yang berjudul “Analisis Perfomansi Algoritma *Round Robin* dengan menggunakan Aturan *Most Time Remaining* ”. Dalam penelitian tersebut analisis perfomansi dilakukan dengan sebuah simulasi. Simulasi dibuat dengan menggunakan *software* CPUSS dan bahasa pemrograman C++ serta menggunakan Visual Studio dalam implementasi algoritma dalam CPUSS.

Pengujian dilakukan dengan 2 kasus scenario uji. Pengujian scenario uji yang pertama dilakukan pengujian pengaruh jumlah *I/O bound* dan pengaruh besarnya nilai *quantum* pada performansi simulasi penjadwalan. Pada scenario uji kedua akan diuji pengaruh jumlah proses terhadap performansi algoritma *round robin* dengan menggunakan aturan *most time remaining*. Pada scenario uji yang kedua akan digunakan jumlah proses *I/O bound* dan nilai *quantum* dari hasil pengujian skenario uji pertama yang menghasilkan turn arround time terbaik sebagai standar penilaian performansi penjadwalan yang baik.

Mekanisme penilaian performansi algoritma *round robin* dengan aturan *most time remaining* dibandingkan dengan Linux *time sharing*. Berdasarkan pengujian tersebut *round robin* dengan aturan *most time remaining* tidak lebih baik daripada Linux *time sharing*, hal ini didasarkan pada perbandingan parameter-parameter uji seperti *waiting time, turn arround time, dan response time.*

Penelitian ini dibuat berdasarkan penelitian yang pernah dilakukan oleh

Boseren (2016) yang berjudul “Analisis Perbandingan Algoritma Penjadwalan *CPU A New Improved Round Robin* dan *A Dynamic Time Quatum Shortest Job*

*Round Robin* ”. Penelitian ini dimaksudkan untuk membandingkan dua algoritma penjadwalan CPU yang berdasarkan pada algoritma *round robin* yaitu *A New Improved Round Robin* (NIRR) dan *A Dynamic Time Quatum Shortest Job Round Robin* (SJRR) dengan berdasarkan pada kriteria *average waiting time* (AWT)*, average turnaround time* (ATAT)*, average response time* (ART)*, number of context switches* (NCS)*, throughput, fairness,* dan CPU *utilization.*

Penelitian dilakukan dengan memberikan *burst time* proses secara acak, proses *id*, *quantum time*, dan *arrival time* ditentukan sesuai dengan *policy* kedua algoritma. *Utilization* kedua algoritma mencapai 100% karena *processor* tidak mengalami kondisi *busy waiting*, *throughput* kedua algoritma bernilai 4, atau 4 proses selesai dikerjakan dalam 25ms, 25 ms ialah nilai tengah dari total *burst time* proses. *Fairnes* bernilai *yes* sesuai dengan *policy* algoritma menetapkan *quantum time*. Nilai-nilai *average waiting time* (AWT)*, average turnaround time* (ATAT)*, average response time* (ART) didapatkan dengan menjumlahkan nilai *waiting time, turnaround time, dan response time* setiap proses diolah hingga selesai dijumlahkan dan dibagi dengan banyaknya proses, sedangkan *number of context switches* merupakan jumlah perpindahan pengolahan CPU terhadap proses sesuai dengan penghitungan *quantum time* masing-masing algoritma, menghasilkan selisih waktu *average waiting time* 3 ms, *average turnaround time* 1 ms, *average response time* 13.8 ms, bagi keunggulan NIRR sedangkan *number of context switches, utilization, throughput,* dan *fairness* bernilai sama yaitu 100%, 6, 4 dan *yes*. Berdasarkan analisis perbandingan yang dilakukan algoritma penjadwalan NIRR lebih optimal jika dibandingkan dengan *A Dynamic Time Quantum* SJRR.

Penelitian ini dibuat berdasarkan penelitian yang dilakukan Indria, Irawati, dan Negara (2013) yang berjudul “Analisis Perbandingan Algoritma Penjadwalan *Deficit Round Robin* (DRR) dan *Smoothed Round Robin* (SRR) pada *Protokol Routing Ad Hoc Demand Distance Vector* (AODV) dan *Destination Sequence*

*Distance Vector* (DSDV) di Jaringan *Mobile Ad Hoc* (MANET)”. Tugas akhir ini menganalisis pengaruh algoritma penjadwalan pada MANET. Algoritma penjadwalan yang digunakan dalam simulasi adalah algoritma penjadwalan *Deficit Round Robin* (DRR) dan *Smoothed Round Robin* (SRR)*,* dimana keduanya menggunakan *Protokol Routing Ad Hoc Demand Distance Vector* (AODV) dan *Destination Sequence Distance Vector* (DSDV)dengan menggunakan *software* NS-2.34 pada layanan *triple play* berupa data, video, dan suara.

Dari penelitian ini didapatkan hasil perbandingan performansi kedua algoritma penjadwalan SRR dan DRR pada routing AODV dan DSDV. Algoritma SRR memberikan performansi lebih baik untuk layanan data dan suara, dan algoritma DRR memiliki performansi lebih baik untuk layanan video. Sedangkan untuk kasus *background traffic* performansi algoritma SRR lebih baik untuk ketiga layanan dibandingkan DRR. Performansi yang baik dihasilkan oleh algoritma SRR dan DRR yang pada routing AODV, dimana nilai *packet loss* yang dihasilkan adalah 0, untuk jumlah node sampai 20 dalam keadaan diam dan kecepatan 1 m/s. Untuk kasus panjang *buffer*, algoritma SRR lebih baik untuk layanan data dan suara dan algoritma DRR lebih baik untuk video. Pada kasus *fairness* *index*, layanan data memiliki *fairness index* lebih kecil dibanding video dan suara untuk kedua algoritma tersebut.

## BAB II LANDASAN TEORI

**2.1 Sistem Operasi**

Secara umum, sistem operasi merupakan *software* pada lapisan pertama yang diletakkan pada memori komputer pada saat komputer dinyalakan. Sedangkan *software-software* lainnya dijalankan setelah sistem operasi berjalan. Sistem operasi akan melakukan layanan inti umum untuk *software-software* itu seperti akses ke *disk,* manajemen memori, *schedule task,* dan antar muka user (*system call*). Bagian kode yang melakukan tugas-tugas inti dan umum tersebut dinamakan *kernel*.

Sistem operasi adalah perangkat lunak sistem yang bertugas untuk melakukan kontrol dan manajemen perangkat keras serta operasi-operasi dasar sistem, termasuk memberikan sumber daya pada program yang sedang beroperasi. Sistem operasi berfungsi sebagai penghubung antara lapisan *hardware* dan *software*, melakukan semua perintah penting dalam komputer, serta menjamin aplikasi-aplikasi yang berbeda fungsinya dapat berjalan lancar secara bersamaan.

Sistem operasi menjamin aplikasi perangkat lunak lainnya agar dapat memakai memori, melakukan *input* serta *output* terhadap peralatan lain, dan mempunyai akses kepada sistem file. Jika beberapa aplikasi berjalan secara bersamaan, maka sistem operasi akan mengatur penjadwalan yang tepat, sehingga semua proses pada komputer yang berjalan mendapatkan waktu yang cukup untuk menggunakan CPU dan tidak menggangu kinerja dari perangkat lain.

**2.2 Sejarah Sistem Operasi**

Menurut Tanenbaum (2001), sistem operasi mengalami perkembangan yang sangat pesat, yang dapat dibagi ke dalam 4 generasi :

1. Generasi Pertama (1945-1955)

Generasi pertama merupakan awal perkembangan sistem komputasi elektronik sebagai pengganti sistem komputasi mekanik, hal itu disebabkan kecepatan manusia untuk menghitung terbatas dan manusia sangat mudah untuk

8

membuat kecerobohan, kekeliruan bahkan kesalahan. Pada generasi ini belum ada sistem operasi, maka sistem komputer diberi instruksi yang harus dikerjakan secara langsung. 2. Generasi Kedua (1955-1965)

Generasi kedua memperkenalkan *Batch Processing System*, yaitu *job* yang dikerjakan dalam satu rangkaian, lalu dieksekusi secara berurutan. Pada generasi ini sistem komputer belum dilengkapi sistem operasi, tetapi beberapa fungsi sistem operasi telah ada, contohnya fungsi sistem operasi ialah FMS dan IBSYS.

3. Generasi Ketiga (1965-1980)

Pada generasi ini perkembangan sistem operasi dikembangkan untuk melayani banyak pemakai sekaligus, dimana para pemakai interaktif berkomunikasi lewat terminal secara on-line ke komputer, maka sistem operasi menjadi *multiuser* (di gunakan banyak pengguna sekaligus) dan *multi-programming* (melayani banyak program sekaligus). 4. Generasi Keempat (Pasca 1980an)

Dewasa ini, sistem operasi dipergunakan untuk jaringan komputer dimana pemakai menyadari keberadaan komputer-komputer yang saling terhubung satu sama lainnya. Pada masa ini para pengguna juga telah dinyamankan dengan *graphical user interface* yaitu antar-muka komputer yang berbasis grafis yang sangat nyaman, pada masa ini dimulai era komputasi tersebar dimana komputasi-komputasi tidak lagi berpusat di satu titik, tetapi dipecah dibanyak komputer sehingga tercapai kinerja yang lebih baik.

**2.3 Proses**

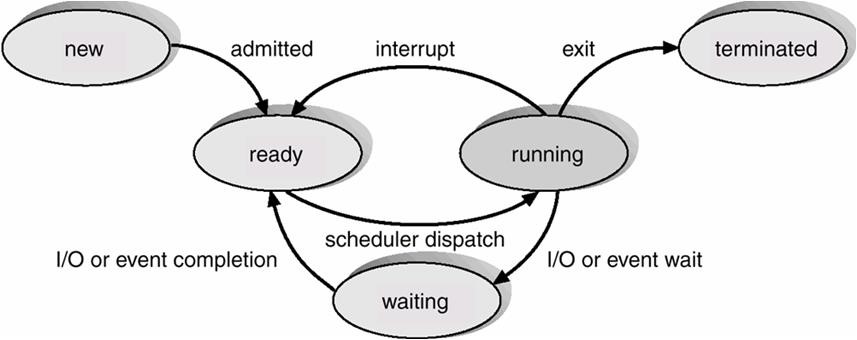
Proses adalah sebuah program yang dieksekusi yang mencangkup *program counter*, *register*, dan variabel didalamnya (Tanenbaum 2001). Secara informal, proses adalah program yang sedang dieksekusi. Suatu proses adalah lebih dari kode program, dimana kadang kala proses dikenal sebagai bagian tulisan. Proses juga termasuk aktivitas yang sedang terjadi, sebagaimana digambarkan oleh nilai pada program *counter* dan isi dari daftar *processor register*. Suatu proses umumnya juga termasuk *proses stack*, yang berisikan data *temporary* (seperti parameter metode, *address* yang kembali, dan variabel lokal) dan sebuah *data section*, yang berisikan variabel global. Sebagaimana proses bekerja, maka proses tersebut merubah state (keadaan statis/asal). Status dari sebuah proses didefenisikan dalam bagian oleh aktivitas yang ada pada proses tersebut (Lumbantoruan 2016).

Konsep kunci dalam semua sistem operasi adalah proses. Sebuah proses pada dasarnya adalah program yang dieksekusi. Proses terkait dengan *address space* atau daftar lokasi memori mulai dari ruang 0 sampai keruang maksimum, dan proses dapat membaca dan menulis pada *address space* tersebut. *Address space* berisi program yang dieksekusi, data dan *stack*. Proses membutuhkan sejumlah sumber daya umumnya termasuk *register* ( *program counter* dan *stack pointer* ), file, proses lain yang terkait dengan proses tersebut, dan semua informasi lainnya yang dibutuhkan untuk menjalankan program tersebut. Proses menampung semua informasi yang dibutuhkan untuk mengeksekusi program (Lumbantoruan 2016).

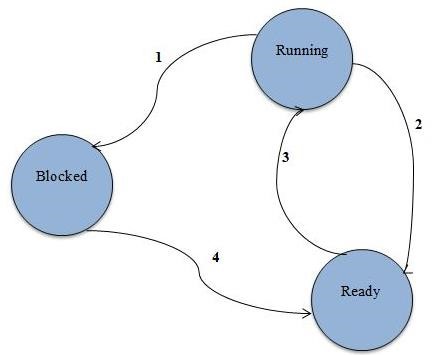
Sebagaimana proses bekerja, maka proses tersebut merubah state (keadaan statis/asal). Tiap-tiap proses terdiri atas suatu keadaan yang terpisah, namun adakalanya proses-proses tersebut butuh untuk saling berinteraksi. Satu proses dapat dibangkitkan dari output proses lainnya sebagai input. Status dari sebuah keadaan proses didefenisikan dalam bagian oleh aktivitas yang ada dari proses tersebut. Diagram state proses terdiri dari 5 keadaan yaitu :

1. *New*, status yang dimiliki pada saat proses baru saja dibuat.
2. *Running*, status yang dimiliki pada saat intruksi-intruksi dari sebuah proses dieksekusi.
3. *Waiting*, status yang dimilliki pada saat proses menunggu suatu event (contohnya: proses *I/O*).
4. *Ready*, status yang dimilki pada saat proses siap untuk dieksekusi oleh *processor.*

1. *Terminated*, status yang dimilki pada saat proses telah selesai dieksekusi. (Lumbantoruan 2016)



## Gambar 2.1 Keadaan Proses



## Gambar 2.2 Diagram Proses States

1. *Process blocks for input*

Terjadi pada saat sebuah proses menemukan bahwa dirinya tidak bisa dilanjutkan, pada beberapa sistem, proses ini harus melaksanakan sistem *call*, *blocked*, pada sistem yang lain pada saat proses tidak mendapatkan yang diinginkan, secara otomatis akan masuk pada sistem *blocked.*

1. *Scheduler picks another process*

*Scheduler* memutuskan bahwa sebuah proses sudah berjalan terlalu lama dan sudah waktunya untuk mengambil proses lain.

1. *Scheduler picks this process*

Proses-proses yang lain sudah mendapatka jatahnya, dan tiba giliran proses lain yang tertunda.

1. *Input becomes available*

Kejadian diluar yang sedang ditunggu sebuah proses sudah terlaksana, misal sebuah proses yang menunggu input dari sebuah output proses yang lain. Bila tidak ada proses yang berjalan , maka transisi 3 segera dilaksanakan dan proses lain segera berjalan, atau menunggu di *ready queue* sampai CPU tersedia.

**2.4 Operasi Pada Proses**

Sistem operasi dalam mengelola proses dapat melakukan operasi-operasi terhadap proses. Operasi-operasi yang dilakukan terhadap proses antara lain:

* Penciptaan proses,
* Terminasi proses,
* Penundaan proses,
* Pelanjutan proses kembali,
* Pengubahan prioritas proses,
* Mem-*block* proses,
* Membangkitkan proses,
* Menjadwalkan proses,
* Proses berkomunikasi dengan proses lain.

**2.4.1 Penciptaan Proses**

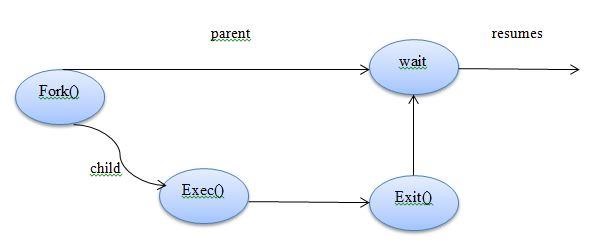
Ada beberapa aktivitas berkenaan dengan pembuatan proses, antara lain:

1. Memberi identitas (nama) pada proses yang dibuat;
2. Menyisipkan proses pada list proses atau tabel proses;
3. Menentukan prioritas awal proses;
4. Membuat PCB;
5. Mengalokasikan *resource* awal bagi proses tersebut.

Ada beberapa kejadian yang menyebabkan pembuatan suatu proses baru,

antara lain;

1. Pada lingkungan *bacth* sebagai tambahan atas pemberian job. Setelah menciptakan proses baru, sistem operasi melanjutkan untuk membaca job selanjutnya.
2. Pada lingkungan interaktif, pada saat *user* baru saja *log in*;
3. Sebagai tanggapan atas suatu aplikasi (seperti mencetak file, sistem operasi dapat menciptakan proses yang akan mengelola percetakan itu); 4) Proses menciptakan proses lain (*child*).



## Gambar 2.3 Proses Penciptaan Proses

Selama eksekusi, suatu proses mungkin akan membuat suatu proses yang baru. Proses tersebut dinamakan *parent*, sedangkan proses yang dibuat dinamakan *child*. Proses pembuatan proses anak membentuk pohon proses. Pembagian sumber daya:

1. *Parent* dan *child* membagi semua sumber daya yang ada.
2. *Child* menggunakan sebagian dari sumber daya yang digunakan *parent*.
3. *Parent* dan *child* tidak membagi sumber daya.

Bentuk eksekusi:

1. *Parent* melanjutkan eksekusi beriringan dengan *children*.
2. *Parent* menunggu hingga beberapa atau seluruh *children* selesai.

Bentuk ruang alamat:

1. *Child* adalah duplikat dari proses tersebut.
2. *Child* mempunyai program yang diambil dari dirinya.

**2.4.2 Penghentian Proses**

Suatu proses berhenti jika telah menyelesaikan pernyataan terakhir, dan meminta pada sistem operasi untuk menghapusnya dengan menggunakan *system call exit*. Proses mengembalikan semua data (*output*) ke *parent* proses melalui *system call wait*. Kemudian proses dihapus dari *list* atau tabel sistem, dilanjutkan dengan menghapus PCB. *Parent*  dapat menghentikan eksekusi proses *child*  dengan menggunakan *system call abort*. Proses anak dihentikan *parent* karena beberapa alasan, antara lain:

* *Child* mengalokasikan sumber daya yang melampaui batas.
* Tugas *child* tidak dibutuhkan lebih lanjut.
* *Parent* berhenti, karena sistem operasi tidak mengizinkan *child* untuk melanjutkan jika *parent* berhenti dan terminasi dilanjutkan.

Ada 14 kejadian yang dapat mengakibatkan suatu proses dihentikan:

1. Selesaikan proses secara normal: proses mengeksekusi *system call* sistem operasi untuk menandakan bahwa proses telah berjalan secara lengkap.
2. Melebihi batas waktu: proses berjalan lama dari batas waktu yang diberikan, terdapat beberapa kemungkinan untuk jenis ukuran waktu. Hal ini termasuk waktu yang telah dilalui, jumlah waktu yang dibutuhkan untuk mengeksekusi, dan jumlah waktu ketika *user* terakhir kali member *input*.
3. Memori tidak tersedi: proses meminta memori yang lebih besar daripada yang disediakan oleh sistem.
4. Pelanggaran terhadap batas memori: proses mencoba meminta untuk mengakses lokasi memori yang tidak diizinkan, atau kata lain proses melanggar keamanan sistem.
5. Kesalahan proteksi: proses berusaha menggunakan sumber daya atau file yang tidak diizinkan dipakainya, atau proses mencoba menggunakannya tidak untuk peruntukannya, seperti menulis file *read-only*. Proses melanggar keamanan sistem.
6. Kesalahan aritmatika: proses mencoba komputasi terlarang. Seperti pembagian dengan nol atau mencoba untuk menyimpan nilai numeric yang lebih besar dari yang dapat ditampung oleh *hardware*.
7. Waktu telah kadaluwarsa: proses mencoba komputasi yang terlarang. Seperti pembagian dengan nol atau mencoba untuk menyimpan nilai numeric yang lebih besar dari yang dapat ditampung oleh *hardware*.
8. Kegagalan I/O: kesalahan muncul pada I/O, seperti ketidakmampuan menemukan file, kegagalan membaca atau menulis setelah sejumlah maksimum percobaan (ketika, misalnya area rusak terjadi pada tape atau operasi tidak sah seperti membaca line dari *line printer*).
9. Cacat intruksi: proses berusaha untuk mengeksekusi intruksi yang tidak ada.
10. Usaha memakai intruksi yang tak diizinkan: proses berusaha untuk menggunakan intruksi yang dicadangkan untuk sistem operasi.
11. Kesalahan menggunakan data: bagian data adalah tipe yang salah atau tidak diinisialisasi.
12. Campur tangan operator atau sistem operasi: untuk suatu alasan, operator atau sistem operasi mengakhiri proses (misalnya, jika terdapat *deadlock*).
13. Berakhirnya proses induk: ketika proses induk berakhir, pada sistem operasi untuk dirancang secara otomatis mengakhiri semua anak proses dari proses induk.
14. Permintaan proses-proses induk mempunyai otoritas untuk menghentikan sembarang proses anaknya.

**2.5 Manajemen Proses**

Proses adalah sebuah program yang sedang dieksekusi. Sedangkan program adalah kumpulan intruksi yang ditulis ke dalam bahasa yang dimengerti sistem operasi. Sebuah proses membutuhkan sejumlah sumber daya untuk menyelesaikan tugasnya. Sumber daya tersebut dapat berupa CPU *time*, alamat memori, berkas-berkas, dan perangkat-perangkat. Sistem operasi mengalokasikan sumber-sumber daya tersebut saat proses itu sedang dijalankan. Ketika prosesproses tersebut berhenti dijalankan, sistem operasi akan mengambil kendali semua sumber daya agar bisa digunakan kembali oleh proses lainnya.

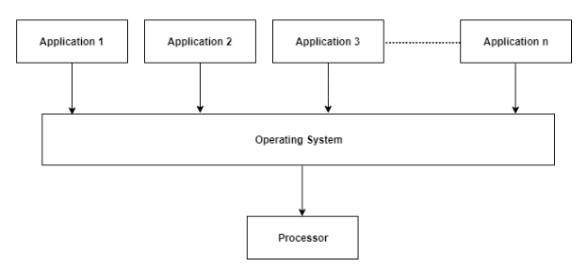
Sistem operasi bertanggung jawab atas aktivitas-aktivitas yang berkaitan dengan manajemen proses seperti :

1. ***Membuat dan menghapus proses pengguna dan sistem proses.*** Sistem operasi bertugas mengalokasikan sumber daya yang dibutuhkan oleh sebuah proses dan kemudian mengambil sumber daya itu kembali setelah proses tersebut selesai agar dapat digunakan untuk proses lainnya.
2. ***Menunda atau melanjutkan proses.*** Sistem operasi akan mengatur proses apa yang harus dijalankan terlebih dahulu berdasarkan prioritas dari proses-proses yang ada. Apabila terjadi 2 atau leibh proses yang mengantri untuk dijalankan, sistem operasi akan mendahulukan proses yang memiliki prioritas paling besar.
3. ***Menyediakan mekanisme untuk proses sinkronisasi.*** Sistem operasi akan mengatur jalannya beberapa proses yang dieksekusi bersamaan. Tujuannya adalah menghidarkan terjadinya inkonsistensi data karena akses data yang sama, juga untuk mengatur urutan jalannya proses agar setiap proses berjalan dengan lancar.
4. ***Menyediakan mekanisme untuk proses komunikasi.*** Sistem operasi menyediakan agar beberapa proses dapat saling berinteraksi dan berkomunikasi ( contohnya sebagai sumber daya antar proses ) satu sama lain tanpa menyebabkan terganggunya proses yang lainnya.
5. ***Menyediakan mekanisme untuk penanganan deadlock****.* *Deadlock* adalah suatu keadaan dimana sistem seperti terhenti karena setiap proses memilki sumber daya yang tidak bisa dibagi dan menunggu untuk mendapatkan sumber daya yang sedang dimiliki oleh proses lain. Saling menunggu inilah yang disebut *deadlock* ( kebuntuan ). Sistem operasi harus bisa mencegah, menghindari, dan mendeteksi adanya *deadlock*. Jika *deadlock* terjadi, sistem operasi juga harus dapat memulihkan kondisi sistemnya.

Manajemen proses merupakan konsep pokok dalam sistem operasi, sehingga masalah manajemen proses adalah masalah utama dalam perancangan sistem operasi. Proses dapat juga didefenisikan sebagai unit kerja terkecil yang secara individu memiliki sumber daya dan dijadwalkan oleh sistem operasi.

### 2.6 Single Processing

*Single Processing* merupakan sistem prosesor yang terdiri satu prosesor. Pada *single processing* hanya terdapat satu proses yang dapat dieksekusi pada suatu waktu yang dipilih dari antrian proses yang telah siap. Sebagian besar komputer yang biasa digunakan oleh user umumnya hanya mengandung sistem dengan satu prosesor/prosesor tunggal.



### Gambar 2.4 Single Processing System

Seperti pada gambar diatas, ada beberapa aplikasi yang perlu dieksekusi. Namun, sistem berisi prosesor tunggal dan hanya satu proses yang dapat dieksekusi pada suatu waktu.

**2.7 Penjadwalan Proses**

Menurut Tanenbaum (2001), penjadwalan proses merupakan kumpulan kebijaksanaan dan mekanisme di sistem operasi yang berkaitan dengan urutan kerja yang dilakukan sistem komputer.

Adapun penjadwalan bertugas memutuskan :

1. Proses yang berjalan.
2. Kapan dan selama berapa lama proses itu berjalan.

Menurut Tarore (2012), pengaturan waktu atau penjadwalan dari kegiatankegiatan yang terlibat didalamnya dimaksudkan agar suatu proyek dapat berjalan dengan lancar serta efektif. Oleh karena itu, pelaksana dari suatu kegiatan biasanya membuat suatu jadwal waktu kegiatan atau *time schedule*. Jadwal waktu kegiatan adalah urutan-urutan kerja yang berisi tentang :

1. Jenis pekerjaan yang akan diselesaikan.
2. Waktu bilamana suatu pekerjaan dimulai dan diakhiri.

Oleh karena itu penjadwalan yang baik harus memiliki ukuran agar prosesproses yang dijalankan lebih optimal. Untuk mengukur dan optimasi kinerja penjadwalan. Menurut Tarek (2006) bahwa ada beberapa hal yang perlu diperhatikan .

1. Adil (*Fairness*)

Adalah proses-proses yang di perlakukan sama, yaitu mendapat jatah waktu pemroses yang sama dan tidak ada proses yang tidak mendapat layanan pemroses sehingga mengalami kekurangan waktu.

1. Efisiensi (*Efficiency*)

Efisiensi atau utilasi pemroses dihitung dengan perbandingan (rasio) waktu sibuk pemroses.

1. Waktu tanggap (*Response time*)

Waktu tanggap adalah waktu yang dibutuhkan untuk merespon atau menanggapi permintaan layanan eksekusi dari sebuah proses. Waktu tanggap dibedakan atas dua hal yaitu :

* + 1. Sistem interaktif

Didefinisikan sebagai waktu yang dihabiskan dari saat karakter terakhir dari perintah dimasukkan atau transaksi sampai hasil pertama muncul di layar. Waktu tanggap ini disebut terminal *response time*.

* + 1. Sistem waktu nyata

Didefinisikan sebagai waktu dari saat kejadian (internal atau eksternal) samapai intruksi pertama rutin layanan yang dimaksud dieksekusi, disebut *event response time.*

1. *Turnaround time*

Adalah waktu yang dihabiskan dari saat program atau *job* mulai masuk ke sistem sampai proses diselesaikan sistem. Waktu yang dimaksud adalah waktu yang dihabiskan di dalam sistem, diekspresikan sebagai penjumlah waktu eksekusi (waktu pelayanan *job*) dan waktu tanggap menunggu, yaitu ***turnaround time* = waktu eksekusi + waktu menunggu.**

1. *Throughput*

Adalah jumlah kerja yang dapat diselesaikan dalam satu unit waktu. Cara untuk mengekspresikan *throughput* adalah dengan jumlah job pemakai yang dapat dieksekusi dalam satu unit/interval waktu.

Tujuan penjadwalan proses adalah untuk meminimalkan total biaya layanan komputer dan waktu tunggu *user*. Dalam prakteknya ada dua masalah yang sering dihadapi, waktu pelayanan terhadap proses dapat dikurangi dengan memperhatikan waktu *processor* hilang akibat intervensi *user*, perangkat keras yang lambat, dan *multiplexing* dari sumber daya. Hal ini memiliki pengaruh yang dratis pada mode operasi yang ditawarkan pada *user* (Lumbantoruan 2016).

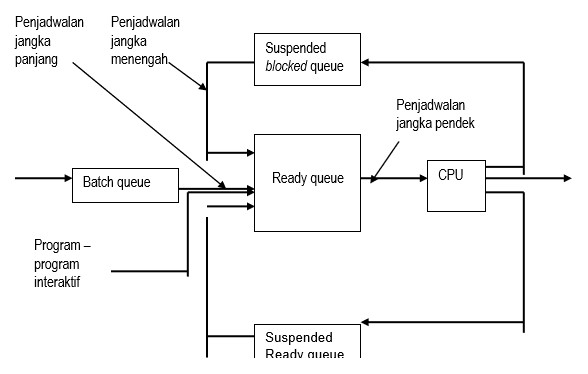
**2.8 Tipe Penjadwalan**

Menurut Hariyanto (2009), terdapat tiga tipe penjadwalan berada secara bersama-sama pada sistem operasi yang kompleks, yaitu :

1. Penjadwal jangka pendek (*Short term scheduller*)
2. Penjadwal jangka menengah (*Medium term scheduller* )
3. Penjadwal jangka panjang (*long term scheduller*)

Sasaran penjadwalan berdasarkan tipe-tipe penjadwalan antara lain :

1. Memaksimumkan kinerja untuk memenuhi satu kumpulan kriteria yang diharapakan.
2. Mengendalikan transisi dari *suspended to ready* (keadaan *suspend* ke *ready*) dari proses *swapping*. *Swapping* adalah pemindahan proses yang tertunda dari memori utama ke memori sekunder.
3. Memberi keseimbangan *job-job* campuran.



## Gambar 2.5 Tipe Penjadwalan Proses

**2.9 Strategi Penjadwalan**

Menurut Hariyanto (2009), terdapat dua strategi penjadwalan, yaitu :

1. Penjadwalan *non preemptive* (*run to completion*)

Proses diberi jatah waktu oleh pemroses, maka pemroses tidak dapat diambil alih oleh proses lain sampai proses itu selesai.

1. Penjadwalan *preemptive*

Proses diberi jatah waktu oleh pemroses, maka pemroses dapat diambil alih proses lain, sehingga proses disela sebelum selesai dan harus dilanjutkan menunggu jatah waktu pemroses tiba kembali pada proses itu.

**2.10 Algoritma-Algoritma Penjadwalan**

Ada beberapa jenis-jenis algoritma penjadwalan. Algoritma penjadwalan dibagi dalam dua konsep secara umum yaitu :

1. *Non-preemptive*, menggunakan konsep :
   1. FIFO (*First In First Out*) atau FCFS (*First Come First Serve*)
   2. SJF (*Shortest Job First*)
   3. HRN (*Highest Ratio Next*)
   4. MFQ (*Multiple Feedback Queues*)
2. *Preemptive*, menggunakan konsep :
   1. RR (*Round Robin*)
   2. SRF (*Shortest Remaining First*)
   3. PS (*Priority Schedulling*)
   4. GS (*Guaranteed Schedulling*)

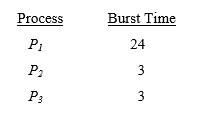
Klasifikasi lain, selain berdasarkan dapat atau tidaknya suatu proses diambil secara paksa adalah klasifikasi berdasarkan adanya prioritas di prosesproses tersebut, yaitu algoritma penjadwalan tanpa berprioritas dan algoritma penjadwalan berprioritas. Algoritma penjadwalan berprioritas ada yang bersifat statis dan ada yang bersifat dinamis (Gea 2015).

**2.11 Algoritma *Round Robin***

Konsep dasar dari algoritma *round robin* adalah dengan menggunakan *time-sharing.* Pada dasarnya algoritma ini sama dengan FCFS, hanya saja bersifat preemptive. Setiap proses mendapatkan waktu CPU yang disebut dengan waktu kuantum (*quantum time*) untuk membatasi waktu proses, biasanya 1-100 milidetik (Gea 2015).

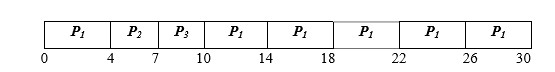
Setelah waktu habis, proses ditunda dan ditambahkan pada *ready queue.* Jika suatu proses memiliki CPU *burst* lebih kecil dibandingkan dengan waktu kuantum, maka proses tersebut akan melepaskan CPU jika telah selesai bekerja, sehingga CPU dapat segera digunakan oleh proses selanjutnya. Sebaliknya, jika suatu proses memilki CPU *burst* yang lebih besar dibandingkan dengan waktu kuantum, maka proses tersebut akan dihentikan sementara jika sudah mencapai waktu kuantum, dan selanjutnya mengantri kembali pada posisi ekor dari *ready queue*, CPU kemudian menjalankan proses berikutnya (Gea 2015). Menurut Tanenbaum (2001), jika terdapat *n* proses pada *ready queue* dan waktu kuantum *q*, maka setiap proses mendapatkan 1/*n* dari waktu CPU paling banyak *q* unit waktu pada sekali penjadwalan CPU. Tidak ada proses yang menunggu lebih dari (*n*-1)*q* unit waktu.

Perfomansi algoritma *round robin* dapat dijelaskan sebagai berikut, jika *q* besar, maka yang digunakan adalah algoritma FIFO, tetapi jika *q* kecil maka sering terjadi *context switch*. Misalkan ada 3 proses : *p1*, *p2*, dan *p3* yang meminta pelayanan CPU dengan *quantum time* sebesar 4 milidetik.



## Gambar 2.6 Contoh Proses

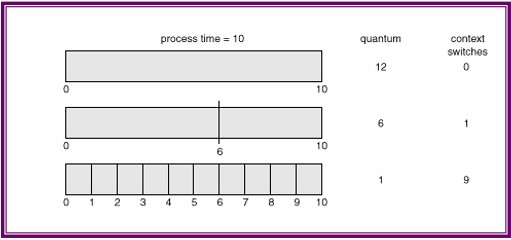
Penjadwalan proses dengan algoritma *round robin* dapat dilihat pada *gantt chart* berikut :



### Gambar 2.7 Gantt Chart

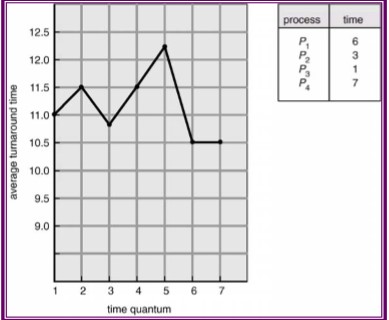
Waktu tunggu untuk *P1* adalah 6, *P2* adalah 4, dan *P3* adalah 7 sehingga rata-rata waktu tunggu adalah (6 + 4 + 7) / 3 = 5.66 milidetik.

Algoritma *round robin* di satu sisi memiliki keuntungan, yaitu adanya keseragaman waktu. Namun di sisi lain, algoritma ini akan terlalu sering melakukan *switching* seperti yang terlihat pada Gambar 2.6. Semakin besar *kuantum-time*nya maka *switching* yang terjadi akan semakin sedikit.



**Gambar 2.8 Menunjukan Waktu Kuantum lebih kecil Meningkatkan *Context Switch***

Waktu *turnaround* juga tergantung ukuran waktu *quantum*. Seperti pada Gambar 2.7, rata-rata waktu *turnaround* tidak meningkat bila waktu *quantum* dinaikkan. Secara umum, rata-rata waktu *turnaround* dapat ditingkatkan jika banyak proses menyelesaikan CPU *burst* berikutnya sebagai satu waktu *quantum*. Sebagai contoh, terdapat tiga proses masing-masing 10 unit waktu dan waktu *quantum* 1 unit waktu, rata-rata waktu *turnaround* adalah 29. Jika waktu *quantum* 10, sebaliknya, rata-rata waktu *turnaround* turun menjadi 20.



## Gambar 2.9 Menunjukan Waktu *Turnaround* Berbeda pada Waktu *Quantum* yang Berbeda

### 2.12 Shortest Job First (SJF)

Algoritma *shortest job first* sangat optimal, karena memberikan rata-rata waktu tunggu lebih kecil dibandingkan algoritma penjadwalan yang lain dengan cara memindahkan *job-job* pendek di depan *job-job* yang panjang, sehingga akan mengurangi waktu tunggu (Santika dan Hansun 2014). Pada penjadwalan SJF, proses yang memiliki CPU *burst* paling kecil dilayani terlebih dahulu. Terdapat dua skema penjadwalan SJF yaitu :

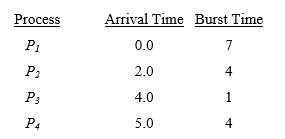
1. *Non-preemptive*

Penjadwalan *non-preemptive* ialah salah satu jenis penjadwalan dimana sistem operasi tidak pernah melakukan *context switch* dari proses yang sedang berjalan ke proses yang lain. Dengan kata lain, proses yang berjalan tidak bisa di *interrupt.* CPU tidak memperbolehkan proses yang ada di *ready queue* untuk menggeser proses yang sedang dieksekusi oleh CPU meskipun proses yang baru tersebut mempunyai *burst time* yang lebih kecil.

1. *Preemptive*

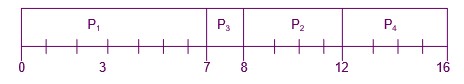
Penjadwalan *Preemptive* mempunyai arti kemampuan-kemampuan sistem operasi untuk memberhentikan proses, sementara proses yang sedang berjalan untuk memberi ruang kepada proses yang prioritasnya lebih tinggi. Jika ada proses yang sedang dieksekusi oleh CPU dan terdapat proses *ready queue* dengan *burst time* lebih kecil daripada proses yang sedang dieksekusi tersebut, maka proses yang sedang dieksekusi oleh CPU akan digantikan oleh proses yang berada di *ready queue* tersebut. *Preemptive* SJF sering disebut juga *shortest-remaining-time-firstscheduling.*

*Shortest job first* adalah algoritma penjadwalan yang optimal dengan ratarata waktu tunggu yang minimal. Misalnya terdapat empat buah proses dengan panjang CPU *burst* dalam milidetik.



## Gambar 2.10 Contoh Proses

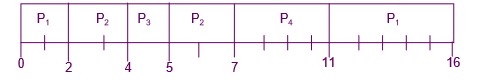
Penjadwalan proses dengan algoritma SJF (*non-preemptive*) dapat dilihat pada gant chart berikut :



### Gambar 2.11 Gantt Chart untuk SJF Non-Preemptive

Waktu tunggu untuk *P1* adalah 0, P2 adalah 26, P3 adalah 3 dan P4 adalah

7 sehingga rata-rata waktu tunggu adalah (0 + 6 + 3 + 7)/4 = 4 milidetik. Sedangkan Penjadwalan proses dengan algoritma SRTF (*preemptive*) dapat dilihat pada gant chart berikut :



## Gambar 2.12 *Gantt Chart* untuk SJF *preemptive*

Waktu tunggu untuk P1 adalah 9, P2 adalah 1, P3 adalah 0 dan P4 adalah 4 sehingga rata-rata waktu tunggu adalah (9 + 1 + 0 + 4)/4 = 3 milidetik.

Meskipun algoritma ini optimal, namun pada kenyataannya sulit untuk diimplementasikan karena sulit untuk mengetahui panjang CPU *burst* berikutnya. Namun nilai ini dapat diprediksi. CPU *burst* berikutnya biasanya diprediksi sebagai suatu rata-rata eksponensial yang ditentukan dari CPU *burst* sebelumnya atau “*Exponential Average*”.

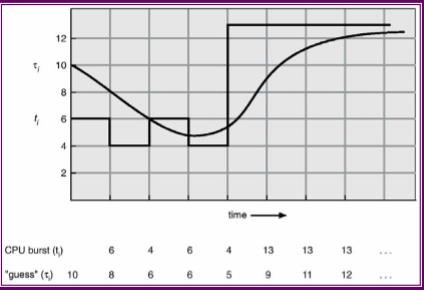


Dengan :

 = Panjang CPU *burst* yang diperkirakan *0* = Panjang CPU *burst* sebelumnya *n* = Panjang CPU *burst* yang ke-*n* (yang sedang berlangsung)

 = ukuran pembanding antara dengan *n* (0 sampai 1)

Grafik hasil prediksi CPU *burst* dapat dilihat pada Gambar 2.10



## Gambar 2.13 : Prediksi Panjang CPU *Burst* Berikutnya

Sebagai contoh , jika  = 0,5 dan :

*CPU burst* (*n*) = 6 4 6 4 13 13 13 … *n* = 10 8 6 6 5 9 11 12 …

Pada awalnya *0 =* 6 dan *n* = 10, sehingga :

2 = 0,5\*6 + (1 – 0,5) \* 10 = 8

Nilai yang digunakan untuk mencari 3 = 0,5 \* 4 + (1 – 0,5) \* 8 = 6

### 2.13 Flowchart

Ada dua *tool* yang sering digunakan untuk membantu menyusun dokumen logika pemrograman, yaitu *flowchart* dan *pseudocode* (kode semu). *Flowchart* adalah simbol-simbol pekerjaan yang menunjukkan bagan aliran proses yang sering terhubung. Jadi, setiap simbol *flowchart* melambangkan pekerjaan dan instruksinya. Simbol-simbol *flowchart* adalah standar yang ditentukan oleh American National Standard Institute Inc.

Fungsi *flowchart* adalah digunakan untuk menganalisa, mendesain, dan mendokumentasikan sebuah proses atau program diberbagai bidang. Secara khusus *flowchart* berfungi untuk menggambarkan situasi apa yang sedang terjadi dan yang akan terjadi dari sebuah simbol dan tanda penghubungnya Selain itu, *flowchart* mampu memperjelas sebuah alur dari suatu sistem baik itu kekurangan atau kelebihan dari berbagai proses didalam tahapan suatu sistem(Syafitri, 2019).

Simbol–simbol yang digunakan dalam *flowchart* ditunjukkan pada Tabel 2.1 berikut:

**Tabel 2.1 Simbol-simbol dalam *Flowchart***

|  |  |  |
| --- | --- | --- |
| **No.** | **Simbol** | **Keterangan** |
| **1.** |  | Simbol *Start* atau *End* yang mendefinisikan awal atau akhir dari sebuah *flowchart.* |
| **2.** |  | Simbol pemrosesan yangterjadi pada sebuah alur kerja. |
| **3.** |  | Simbol *Input/Output* yang mendefinisikan masukkan dan keluaran proses. |
| **4.** |  | Simbol untuk memutuskan proses lanjutan dari kondisi tertentu. |
| **5.** |  | *Storage* menyatakan *input* berasal dari *storage* atau *output* di simpan ke *storage*. |
| **6.** |  | *Preparation* merupakan simbol persiapan yang digunakan untuk memberi nilai awal suatu besaran. |

### 2.14 Unified Modelling Languange (UML)

Unified Modelling Language (UML) adalah sebuah "bahasa" yg telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem piranti lunak. UML menawarkan sebuah standar untuk merancang model sebuah sistem. Dengan menggunakan UML kita dapat membuat model untuk semua jenis aplikasi piranti lunak, dimana aplikasi tersebut dapat berjalan pada piranti keras, sistem operasi dan jaringan apapun, serta ditulis dalam bahasa pemrograman apapun. Tetapi karena UML juga menggunakan class dan operation dalam konsep dasarnya, maka ia lebih cocok untuk penulisan piranti lunak dalam bahasabahasa berorientasi objek seperti C++, Java, C# atau VB.NET. Walaupun demikian, UML tetap dapat digunakan untuk modeling aplikasi prosedural dalam VB atau C.

#### 2.14.1 Use Case Diagram

*Use case diagram* menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah “apa” yang diperbuat sistem, dan bukan “bagaimana”. Sebuah use case merepresentasikan sebuah interaksi antara aktor dengan sistem. *Use case* merupakan sebuah pekerjaan tertentu, misalnya login ke sistem, membuat sebuah daftar belanja, dan sebagainya. Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu.

## Tabel 2.2 Simbol *Use Case Diagram*

|  |  |  |
| --- | --- | --- |
| **No.** | **Simbol** | **Keterangan** |
| **1.** |  | Aktor, Menunjukkan *user* yang akan menggunakan sistem. |
| **2.** |  | *Usecase,* Menunjukkan proses yang terjadi pada sistem. |
| **3.** |  | *Undirectional Association,* Menunjukkan hubungan antara aktor dengan dan *usecase* atau antar *usecase.* |

### 2.14.2 Activity Diagram

*Activity diagram* menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi.

## Tabel 2.3 Simbol *Activity Diagram*

|  |  |  |
| --- | --- | --- |
| **No.** | **Simbol** | **Keterangan** |
| 1. | Start | Kondisi Awal, Menunjukkan awal dari suatu diagram aktivitas. |
| 2. | End | Kondisi Akhir, Menunjukkan akhir dari suatu diagram aktivitas. |
| 3. |  | Kondisi transisi, Menunjukkan kondisi transisi antar aktivitas. |
| 4. |  | Swimlane, Menunjukkan aktor dari diagram aktivitas yang dibuat. |
| 5. |  | Aktivitas, Menunjukkan aktivitas-aktivitas yang terdapat pada diagram aktivitas. |
| 6. |  | Pengecekan kondisi, Menunjukkan pengecekan terhadap suatu kondisi. |

### 2.14.3 Sequence Diagram

*Sequence diagram* menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, *display*, dan sebagainya) berupa message yang digambarkan terhadap waktu. Sequence diagram terdiri atar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait).

**Tabel 2.4 Simbol *Sequence Diagram***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No.** | **Simbol** | | |  | **Keterangan** |
| **1.** |  |  | |  | Objek, Menunjukkan objek yang yang terdapat di diagram *sequence.* |
|  |  |
| **2.** |  | | |  | Pesan ke objek sendiri, Menunjukkan pesan yang diproses pada objek itu sendiri. |
| **3.** |  | | |  | Pesan objek, Menunjukkan pesan yang disampaikan ke objek lain dalam diagram *sequence.* |

## BAB III METODOLOGI PENELITIAN

**3.1 Waktu dan Tempat Penelitian**

**3.1.1 Waktu Penelitian**

Waktu pelaksanaan penelitian tugas akhir dilaksanakan mulai dari bulan Desember 2019 sampai dengan Maret 2020. Rincian kegiatan dapat dilihat pada Tabel 3.1 berikut :

## Tabel 3.1 *Gannt Chart* Waktu Penelitian

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| No | Uraian |  | | | | Waktu (2019-2020) | | | | | | | |  |  | | |
| Desember | | | | Januari | | | | Februari | | | |  | Maret | | |
| 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 |
| 1 | *Analysis* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 | *Design* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 | *Simulation*  *Prototype* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 | *Implementation* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5 | *Monitoring* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

**3.1.2 Tempat Penelitian**

Adapun tempat penelitian tugas akhir akan dilakukan di Lab Multimedia dan Artifcial Intelligence Fakultas Teknik Universitas Halu Oleo.

**3.2 Metode Pengumpulan Data**

Pada penelitian ini, penulis membutuhkan data atau informasi dan referensi yang relatif lengkap. Maka dari itu penulis melakukan riset untuk mendapatkan data atau referensi yang diperlukan. Metode yang digunaka dalam proses pengumpulan data sebagai berikut :

31

32

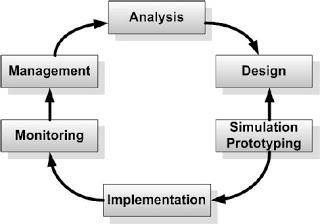
a. Kajian Pustaka

Metode ini digunakan untuk mencari literatur atau sumber pustaka yang berkaitan dengan pengimplementasian *Round Robin* (RR) dan *Shortest Job First* (SJF). Sumber literature berupa buku, *paper*, karya ilmiah, dan beberapa situs ysng dapat menunjang tugas akhir ini.

**3.3 Metode Pengembangan Sistem**

Pada penelitian ini, penulis menggunakan metode *Network Development Life Cycle* (NDLC). Pemelihan metode ini dilakukan dengan alasan bahwa sistem operasi memilki kebutuhan yang berbeda dan memiliki permasalahan sehingga membutuhkan solusi permasalahan yang berbeda dengan melakukan pendekatan yang bervariasi terhadap model NDLC yang dibutuhkan dalam proses pengembangan sistem operasi.

*Network development life cycle* (NDLC) merupakan sebuah metode yang bergantung pada proses pembangunan sebelumnya seperti perencanaan strategi bisnis, daur hidup pengembangan aplikasi, dan analisis pendistribusian data. Adapun skema dari NDLC dapat dilihat pada Gambar 3.1 berikut



## Gambar 3.1 Skema Metode NDLC

1. *Analysis*

Analisis sistem merupakan penguraian dari suatu sistem informasi ke dalam bagian komponennya dengan maksud untuk mengidentifikasikan dan mengevaluasi permasalahan, kesempatan dan hambatan yang terjadi dan kebutuhan yang diharapkan sehingga dapat diusulkan perbaikannya (Masrizal 2017).

Pada tahap ini dilakukan dengan menganalisa permasalahan yang muncul, menganalisa kebutuhan yang akan diterapkan pada sistem.

1. *Design*

Setelah peneliti mendapatkan dokumentasi dari hasil analisis, maka peneliti mengubah kebutuhan-kebutuhan diatas menjadi representasi ke dalam bentuk “*blueprint*” sistem sebelum coding dimulai. Sehingga hasil desain akan digunakan oleh peneliti untuk membangun aplikasi.

1. *Simulation Prototype*

Pada tahap ini peneliti akan melakukan pengujian dengan membuat bentuk simulasi sistem dengan bantuan *tools*.

1. *Implementation*

Untuk dapat dimengerti oleh mesin, dalam hal ini komputer, maka desain sebelumnya harus diubah bentuknya menjadi bentuk yang dimengerti oleh mesin, yaitu kedalam bahasa pemrograman melalui proses *coding.* Tahap ini merupakan implemntasi dari tahap *design* yang secara teknis dikerjakan oleh peneliti.

Pada tahap ini, peneliti membangun sebuah sistem berdasarkan desain “*blueprint*” yang telah dibuat. Pengembangan sistem ini dilakukan dari awal hingga sistem siap dijalankan. Dari fungsi-fungsi yang dibutuhkan hingga tampilan antar muka untuk pengguna.

1. *Monitoring*

Tahapan monitoring merupakan tahapan yang penting, agar sistem yang telah dibuat dapat berjalan sesuai dengan yang diinginkan, maka perlu dilakukan kegiatan *monitoring*.

*Monitoring* dapat berupa melakukan pengamatan pada :

1. Memantau hasil data yang berjalan pada sistem, sudah berjalan sesuai dengan semestinya.
2. Mengukur hasil kinerja penjadwalan *Round Robin* (RR) dan *Shortest Job First* (SJF) dengan melihat *waiting time per process*, *turnaround time per process*, *average waiting time*, dan *average turnaround time* setiap penjadwalan RR dan SJF pada aplikasi simulasi.
3. Evaluasi hasil kinerja penjadwalan *Round Robin* (RR) dan *Shortest Job First* (SJF) pada sistem.

**3.4 Analisis Sistem**

Analisis sistem merupakan suatu tahapan yang bertujuan untuk mengetahui dan mengamati apa saja yang terlibat dalam suatu sistem. Pembahasan yang ada pada analisis sistem ini yaitu analisis masalah dan analisis kebutuhan fungsional.

**3.4.1 Analisis Masalah**

Dalam tugas akhir ini, berdasarkan studi literature dan hasil observasi yang dilakukan, maka dilakukan penelitian analisis perbandingan algoritma penjadwalan *round robin*(RR) dan *shortest job first*(SJF)untuk manajemen proses dalam *single processing.* Percobaan dilakukan pada dua kondisi berbeda yaitu proses yang akan dieksekusi dengan menggunakan algoritma RR dan proses yang akan dieksekusi dengan algoritma SJF. Untuk pengujian penelitian ini menggunakan aplikasi simulasi yang dibuat oleh peneliti.

**4.1.2 Analisis Kebutuhan Nonfungsional**

Analisis kebutuhan nonfungsional adalah suatu langkah dimana peneliti menganalisis sumber daya yang dibutuhkan terhadap penelitina yang akan dilakukan. Analisis kebutuhan nonfungsional yang dilakukan terbagi dalam dua tahap, yaitu kebutuhan perangkat keras dan analisis kebutuhan perangkat lunak.

**4.1.2.1 Analisis Kebutuhan Perangkat Keras**

Perangkat keras(*hardware*) yang digunakan pada penelitian ini adalah satu buah *laptop* sebagai media utama semua kegiatan penelitian. Adapun spesifikasi dari perangkat tersebut dapat dilihat pada Tabel 4.1 berikut:

## Tabel 4.1 Spesifikasi *Laptop*

|  |  |
| --- | --- |
| *Brand* | Acer |
| *Processor* | Intel Core i3 |
| RAM | 4 GB |

34

|  |  |
| --- | --- |
| *Storage Size* | 400 GB |
| *Operation System* | *Windows* 10 |

**4.1.2.2 Analisis Kebutuhan Perangkat Lunak**

Perangkat lunak yang digunakan pada penelitian ini dapat dilihat pada Tabel 4.2 berikut:

## Tabel 4.2 Spesifikasi Perangkat Lunak

|  |  |  |
| --- | --- | --- |
| **No.** | **Nama Perangkat** | **Deskripsi** |
| 1. | *Windows* 10 | *Windows* digunakan untuk menjalankan perangkat lunak yang dibutuhkan dalam penelitian |
| 2. | *Netbeants* IDE 8.2 | *Netbeants* IDE 8.2 digunakan untuk membangun aplikasi simulasi algoritma penjadwalan *round robin* dan *shortest job first* |
| 3. | *Adobe Ilustrator* CC 2018 | *Adobe Ilustrator* CC 2018 digunakan untuk membuat *icon* menu yang akan digunakan pada aplikasi simulasi. |
| 4. | *Mozila Firefox* | *Mozila Firefox* digunakan untuk mengakses *website* informasi yang diperlukan pada penelitian. |

**4.2 Perancangan Sistem**

Perancangan sistem yang akan dibangun terbagi menjadi 2 bagian yaitu perancangan simulasi penjadwalan *round robin*(RR) dan *shortest job first*(SJF)dan analisis perbandingan penjadwalan *round robin*(RR) dan *shortest job first*(SJF)*.*

**4.2.1 Perancangan Simulasi Penjadwalan *Round Robin* (RR) dan *Shortest Job First* (SJF).**

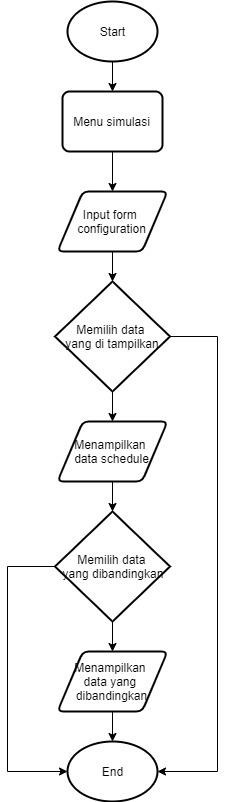
Perancangan simulasi penjadwalan RR dan SJF terdiri atas perancangan

*flowchart*, dan *unified modeling language* (UML) serta perancangan *user interface.*

### 4.2.1.1 Flowchart Sistem

*Flowchart* adalah suatu bagan diagram dengan simbol-simbol grafis yang menyatakan aliran algoritma secara detail dan prosedur sistem secara logika. *Flowchart* memberikan solusi selangkah demi selangkah untuk menyelesaikan masalah yang ada dalam proses. Adapun alur kerja *flowchart* diagram aplikasi adalah sebagai berikut :

1. *User* memilih menu *scenario*, setelah itu mengisi data di form *configuration.*
2. Sistem akan memproses data yang diinputkan oleh *user,* sesuai dengan *rule*/aturan dari algoritma penjadwalan *round robin* dan *shortest job first.*
3. *User* memilih menu *scheduler,* setelah itu memilih data algoritma yang ingin ditampilkan.
4. *User* memilih menu *comparisons*, setelah itu memilih data algoritma yang ingin dibandingkan.



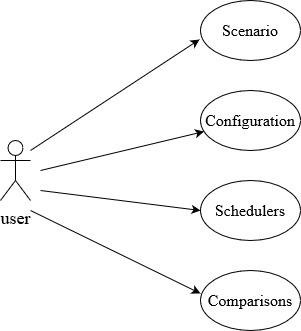
## Gambar 4.1 *Flowchart* Sistem

### 4.2.1.2 Unified Modeling Language ( UML)

Aplikasi dibangun dengan menggunakan *Unified Modeling Language* (UML). UML merupakan bahasa visual untuk pemodelan dan komunikasi mengenai sebuah sistem dengan menggunakan diagram yang terdiri dari *use case diagram, activity diagram,* dan *sequence diagram.*

### 1. Use Case Diagram

*Use Case Diagram* adalah sebuah diagram yang dapat merepresentasikan interaksi yang terjadi antara user dengan sistem. *Use case diagram* mendeskripsikan siapa saja yang menggunakan sistem dan bagaimana cara mereka berinteraksi dengan sistem. *Use case diagram* dari sistem yang akan dibangun dapat dilihat pada Gambar 4.2.



### Gambar 4.2 Use Case Diagram Sistem

Pada gambar 4.2 *use case* *diagram* diatas menggambarkan scenario atau konsep awal dari pembuatan simulasi penjadwalan algoritma *round robin* dan *shortest job first* untuk manajemen proses dalam *single processing.* Pada sistem terdapat *use case* yang saling terhubung dengan *actor.* *Actor* yang berinteraksi langsung dengan aplikasi simulasi ini hanya terdiri dari *user.* Sedangkan untuk *use case* pada sistem ini terdiri atas empat yaitu *scenario*, *configuration*, *schedulers*, dan *comparisons.*

### Tabel 4.3 Main Succes Scenario

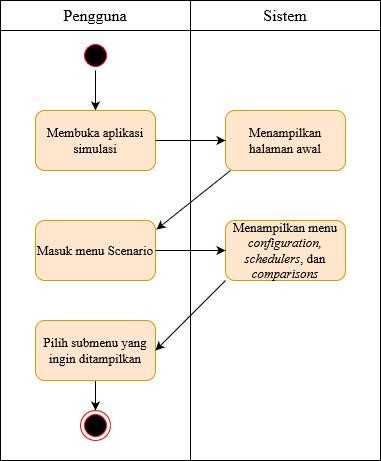
|  |  |
| --- | --- |
| ***User Action*** | ***System Response*** |
| *User* membuka aplikasi simulasi penjadwalan RR dan SJF. | *System* akanmenampilakan halaman utama aplikasi simulasi. |
| *User* memilih menu *scenario.* | *System* akan menampilkan submenu *configuration, schedulers,* dan *comparisons.* |
| *User* memilih menu *schedulers* | *System* akanmenampilkan *form* data algoritma yang ingin ditampilkan. |
| *User* memilih menu *comparisons* | *System* akanmenampilkan *form* data algoritma yang ingin dibandingkan. |

#### 2. Activity Diagram

*Acticity diagram* adalah diagram yang menggambarkan berbagai aliran aktivitas dalam sebuah sistem yang sedang dirancang dan bagaimana masingmasing aliran berawal, keputusan yang mungkin terjadi, dan bagaimana aktivitas tersebut berakhir. Berikut ini adalah *activity diagram* yang akan menggambarkan alir aktivitas sistem.

#### 1. Activity Diagram Scenario

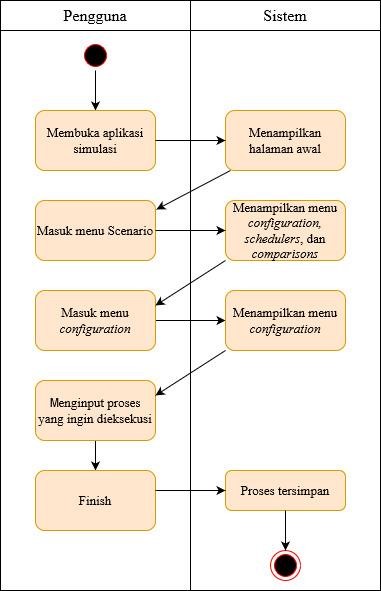
Gambar 4.3 merupakan diagram aktivitas yang menunjukkan aktivitas *user* ketika memilih menu *scenario.*



### Gambar 4.3 Activity Diagram Scenario

#### 2. Activity Diagram Configuration

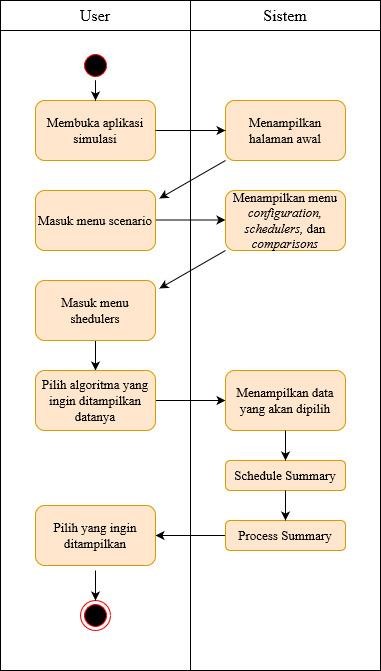
Gambar 4.4 merupakan diagram aktivitas yang menunjukkan aktivitas *user* ketika memilih menu *configuration.*



### Gambar 4.4 Activity Diagram Configuration

#### 3. Activity Diagram Schedulers

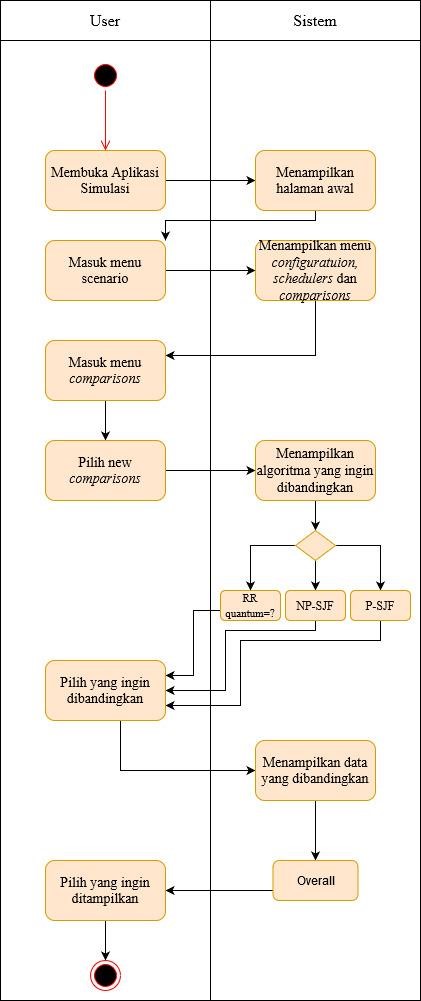
Gambar 4.5 merupakan gambar *Activity Diagram* yang menunjukkan aktivitas *user* memilih menu *schedulers.*



### Gambar 4.5 Activity Diagram Schedulers

#### 4. Activity Diagram Comparisons

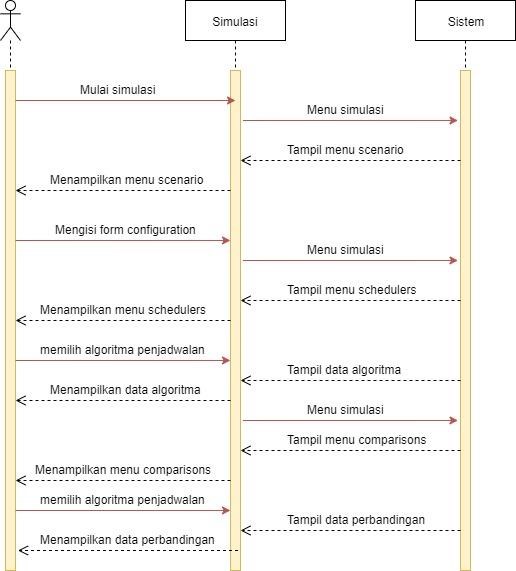
Gambar 4.6 merupakan gambar *Activity Diagram* yang menunjukkan aktivitas *user* memilih menu *comparisons.*



### Gambar 4.6 Activity Diagram Comparisons

#### 3. Sequence Diagram

*Sequence Diagram* menggambarkan interaksi antar objek di dalam dan di sekitar sistem yang digambarkan terhadap waktu. Berikut ini adalah *Sequence Diagram* yang akan menggambarkan interkasi antar objek dan sistem.



### Gambar 4.7 Sequence Diagram

**2.4.2 Analisis Penjadwalan *Round Robin* dan *Shortes Job First***

Setelah aplikasi simulasi penjadwalan *round robin*(RR) dan *shortest job first*(SJF) dijalankan dan diuji. Maka akan dilakukan analisis antara proses yang dijalankan dengan menggunakan RR dan SJF. Analisis dapat dilihat dari perbandingan pengujian proses yang menggunakan RR dan SJF. Untuk parameter dari pengujian yaitu ada dua, pertama nilai *waiting time* dan kedua nilai *turnaround time.* Nilai rata-rata dari *turnaround time* dan *waiting time* akan menjadi kriteria dari pengukuran dalam melakukan pengujian.

## DAFTAR PUSTAKA

Boseren, Paulus V. Daud. 2016. “Analisis Perbandingan Algoritma Penjadwalan

CPU A New Improved Round Robin dan A Dynamic Time Quantum

Shortest Job Round Robin,” no. 672010239.

Gea, Asaziduhu. 2015. “Optimasi Turn Arround Time Pada Penjadwalan Round Robin Dengan Mencari Quantum Time Optimal Menggunakan” 1 (1): 1–9.

Hariyanto, Bambang. 2009. *Sistem Operasi Edisi 1*. Bandung : Informatika.

Indria, Silvi Deby, Indrarini Dyah Irawati, dan Ridha Muldina Negara. 2013. “Analisis Perbandingan Algoritma Penjadwalan Deficit Round Robin ( DRR ) Dan Smoothed Round Robin ( SRR ) Pada Protokol Routing Ad Hoc On

Demand Distance Vector ( AODV ) Dan Destination Sequence Distance Vector ( DSDV ) Di Jaringan Mobile Ad Hoc ( MANET ).”

Kareth, Michael. 2012. “Analisis Optimalisasi Waktu Dan Biaya Dengan Program Primavera 6 . 0 ( Studi Kasus : Proyek Perumahan Puri Kelapa Gading )” 1 (1): 53–59.

Lumbantoruan, Gortap. 2016. “Modifikasi Algoritma Round Robin Dengan Dynamic Quantum Time Dan Pengurutan Proses Secara Ascending” 2 (2): 44–55.

Masrizal. 2017. “Simulasi Perbandingan Penjadwalan Round Robin dan FCFS untuk Manajemen Proses Dalam Single Processing” 1 (1): 36–41.

Santika, Monica, dan Seng Hansun. 2014. “Implementasi Algoritma Shortest Job First dan Round Robin pada Sistem Penjadwalan Pengiriman Barang” VI (2): 94–99.

Sobh, Tarek M, dan Abhilasha Tibrewal. 2006. “Parametric Optimization Of Some Critical Operating System Functions – An Alternative Approach To The Study Of Operating Systems Design,” no. May.

Sonri, Anrio, Endro Ariyanto, Tri Brotoharsono, Fakultas Teknik Informatika, dan

Universitas Telkom. 2013. “Analisis Perfomansi Algoritma Round Robin Dengan Menggunakan Aturan Most Time Remaining.”

Tanenbaum, Andrew S, dan Albert S Woodhull. 2001. *Operating Systems Design and Implementation (3rd Edition)*.

Syafitri, Irmayani.2019."Pengertian Flowchart Beserta Fungsi dan Simbol-Simbol

Flowchart Yang Paling Umum Digunakan." *Nasabamedia.Com* https://www.nasbamedia.com/pengertian-flowchart/Retrieved October 29, 2019