LAPORAN SISTEM OPERASI
MATERI KE – 3
PROSES & THREAD

Dosen Pengampu:
Soffin Nahwa Utama, S.Kom, M.T

OLEH:
Moch Alfan Miftachul Huda
220605110088
KELAS F

LATIHAN SOAL :

1. Sebutkan *state* pada proses dan jelaskan diagram proses
2. Apa yang dimaksud *short term scheduler* dan *long term scheduler* ?
3. Jelaskan 4 alasan mengapa proses harus bekerja sama.
4. Tuliskan kode program untuk penyelesaian permasalahan producer consumer dengan menggunakan *shared memory*.
5. Diketahui skema komunikasi antar proses menggunakan mailbox
   a. Proses *P* ingin menunggu 2 pesan, satu dari mailbox *A* dan satu dari mailbox *B*. Tunjukkan urutan **send** dan **receive** yang dieksekusi
   b. Bagaimana urutan **send** dan **receive** yang dieksekusi *P* jika *P* ingin menunggu satu pesan dari mailbox *A* atau mailbox *B* (salah satu atau keduanya)
6. Jelaskah apa yang dimaksud dengan *thread* dan struktur dari *thread*.
7. Jelaskan empat keuntungan menggunakan *threads* pada *multiple process*.
8. Apakah perbedaan antara *user-level thread* dan *kernel-supported threads* ?
9. Ada 3 model *multithreading*, jelaskan.
10. Jelaskan state pada *Java thread*.

1. **State pada Proses dan Diagram Proses:** Proses dalam sistem operasi dapat berada dalam beberapa state berbeda, yaitu:

   - **New**: Proses baru dibuat tetapi belum siap untuk dieksekusi.
   - **Ready**: Proses siap untuk dijalankan dan menunggu alokasi CPU.
   - **Running**: Proses sedang dijalankan oleh CPU.
   - **Waiting**: Proses menunggu beberapa event eksternal (seperti I/O) untuk selesai.
   - **Terminated**: Proses telah selesai dieksekusi atau dihentikan.

   Diagram proses biasanya digambarkan sebagai diagram keadaan yang menunjukkan transisi antara state-state ini. Misalnya, ketika proses berpindah dari state New ke Ready, atau dari Ready ke Running ketika CPU dialokasikan. Transisi ini juga mencakup perubahan state dari Running ke Waiting ketika proses membutuhkan I/O, dan kembali ke Ready setelah event tersebut selesai. Diagram ini membantu dalam memahami bagaimana proses berpindah antara berbagai keadaan dan bagaimana sistem operasi mengelola eksekusi proses.

2. Short Term Scheduler dan Long Term Scheduler:
    - Short Term Scheduler: Juga dikenal sebagai CPU scheduler, bertanggung jawab untuk memilih proses dari queue Ready dan mengalokasikan CPU. Penjadwalan ini terjadi dalam waktu yang sangat singkat dan sering, memastikan bahwa proses yang siap eksekusi mendapatkan akses CPU secepat mungkin. Tujuan utama dari short term scheduler adalah untuk memaksimalkan penggunaan CPU dan throughput sistem.

    - Long Term Scheduler: Juga dikenal sebagai admission scheduler, bertugas untuk memutuskan proses mana yang akan dimasukkan ke dalam queue Ready dari queue Job. Scheduler ini beroperasi dalam interval yang lebih panjang dan mengatur jumlah proses yang berada di memori utama. Long term scheduler memastikan bahwa sistem tidak terlalu banyak memuat proses, menghindari kelebihan beban memori.

3. Empat Alasan Mengapa Proses Harus Bekerja Sama:
    - Berbagi Data: Proses sering kali perlu bekerja sama untuk berbagi data yang sama, terutama dalam aplikasi yang membutuhkan komunikasi antar proses untuk mencapai tujuan bersama.

    - Koordinasi Tugas: Beberapa tugas mungkin memerlukan koordinasi antara proses untuk menjalankan fungsi yang lebih kompleks atau untuk memastikan hasil yang konsisten.

    - Efisiensi: Dengan bekerja sama, proses dapat meningkatkan efisiensi operasional. Misalnya, satu proses mungkin menangani I/O sementara yang lain menangani kalkulasi, memungkinkan pemanfaatan CPU dan perangkat I/O yang lebih baik.

    - Keamanan dan Integritas: Kerjasama antara proses membantu dalam menjaga keamanan dan integritas sistem dengan memastikan bahwa proses yang saling bergantung dapat beroperasi dengan cara yang terkoordinasi dan aman, mengurangi kemungkinan terjadinya konflik atau kerusakan data.

4. Kode Program Penyelesaian Permasalahan Producer-Consumer dengan Shared Memory (menggunakan Python dan modul multiprocessing sebagai contoh):

```
from multiprocessing import Process, Value, Array, Lock
import time

# Define buffer size
BUFFER_SIZE = 10

def producer(buffer, in_index, out_index, lock):
    for i in range(20):
        time.sleep(1)
        lock.acquire()
        buffer[in_index.value] = i
        print(f"Produced: {i} at index {in_index.value}")
        in_index.value = (in_index.value + 1) % BUFFER_SIZE
        lock.release()

def consumer(buffer, in_index, out_index, lock):
    for _ in range(20):
        time.sleep(2)
        lock.acquire()
        item = buffer[out_index.value]
        print(f"Consumed: {item} from index {out_index.value}")
        out_index.value = (out_index.value + 1) % BUFFER_SIZE
        lock.release()

if __name__ == '__main__':
    # Shared memory setup
    buffer = Array('i', BUFFER_SIZE)  # Shared buffer
    in_index = Value('i', 0)  # Shared index for producer
    out_index = Value('i', 0)  # Shared index for consumer
    lock = Lock()  # Lock for synchronizing access

    # Create producer and consumer processes
    prod = Process(target=producer, args=(buffer, in_index, out_index,
lock))
    cons = Process(target=consumer, args=(buffer, in_index, out_index,
lock))

    # Start processes
    prod.start()
    cons.start()

    # Wait for processes to complete
    prod.join()
cons.join()
```

5. Skema Komunikasi Antar Proses Menggunakan Mailbox:

a. Untuk proses P yang ingin menunggu 2 pesan, satu dari mailbox A dan satu dari mailbox B, urutan send dan receive yang dieksekusi dapat seperti berikut:
    - Receive dari mailbox A: Proses P memulai dengan menerima pesan dari mailbox A. Ini bisa dilakukan dengan melakukan operasi receive untuk mailbox A terlebih dahulu.
    - Receive dari mailbox B: Setelah menerima pesan dari mailbox A, proses P kemudian menunggu dan menerima pesan dari mailbox B.
    - Send ke mailbox A: Jika proses P perlu mengirim pesan ke mailbox A setelah menerima pesan dari mailbox B, ia akan melakukan operasi send ke mailbox A.
    - Send ke mailbox B: Demikian juga, jika diperlukan, proses P dapat mengirim pesan ke mailbox B setelah menerima dari mailbox A.

    Urutan eksekusi operasi send dan receive bisa berbeda tergantung pada implementasi dan kebutuhan spesifik dari komunikasi antar proses, namun pada dasarnya melibatkan menunggu dan menerima pesan dari kedua mailbox yang ditentukan.

b. Urutan Send dan Receive jika P Ingin Menunggu Satu Pesan dari Mailbox A atau Mailbox B: Jika proses P ingin menunggu satu pesan dari mailbox A atau mailbox B (salah satu atau keduanya), proses P biasanya akan menggunakan mekanisme untuk menunggu pesan dari beberapa sumber secara bersamaan. Ini dapat dicapai dengan menggunakan operasi non-blok, seperti select atau poll, atau mekanisme spesifik sistem operasi atau pustaka. Urutannya mungkin sebagai berikut:

    - Setup untuk Menerima Pesan: Proses P mengonfigurasi untuk menunggu pesan dari mailbox A atau mailbox B.

    - Receive dari Mailbox A atau B: Proses P menunggu pesan dari salah satu dari kedua mailbox. Setelah menerima pesan dari salah satu mailbox, proses akan melanjutkan dengan pesan tersebut dan mungkin tidak perlu menunggu pesan dari mailbox yang lain.

    - Proses Pesan: Setelah menerima pesan, proses P akan memproses pesan yang diterima. Jika perlu, proses P dapat melanjutkan dengan operasi lain atau menunggu pesan tambahan.

6. Pengertian dan Struktur Thread:
   Thread adalah unit eksekusi terkecil dalam sebuah proses yang dapat dijadwalkan secara independen. Thread memungkinkan program untuk melakukan beberapa tugas secara bersamaan (concurrency) dalam satu proses.

Struktur Thread meliputi beberapa elemen kunci:

- Program Counter (PC): Menyimpan alamat instruksi berikutnya yang akan dijalankan oleh thread.
- Stack: Menyimpan informasi lokal, termasuk variabel lokal dan alamat pengembalian fungsi.
- Registers: Menyimpan status register CPU khusus untuk thread.
- Thread ID: Identifikasi unik untuk thread.
- Thread State: Status saat ini dari thread (seperti running, waiting, terminated).

7. Keuntungan Menggunakan Threads pada Multiple Process:
    - Efisiensi Sumber Daya: Threads dalam satu proses berbagi ruang alamat dan sumber daya, mengurangi overhead dibandingkan dengan proses yang sepenuhnya terpisah.

    - Responsif: Menggunakan threads memungkinkan aplikasi untuk tetap responsif, karena satu thread bisa menangani input pengguna sementara thread lain mengerjakan tugas latar belakang.

    - Berbagi Data: Threads dapat dengan mudah berbagi data dan status karena mereka berada dalam ruang alamat yang sama, mempermudah komunikasi antar threads.

    - Penggunaan CPU: Threads dapat memanfaatkan multi-core CPU secara lebih efisien dengan memungkinkan eksekusi simultan pada core yang berbeda.

8. Perbedaan antara User-Level Threads dan Kernel-Supported Threads:
    a. User-Level Threads:
        - Dikelola sepenuhnya oleh pustaka thread di level pengguna.
        - Lebih cepat dan efisien dalam hal manajemen thread karena tidak memerlukan interaksi dengan kernel.
        - Tidak dapat memanfaatkan multi-core CPU secara efektif karena kernel hanya mengetahui proses yang berjalan, bukan thread individu di dalamnya.
    b. Kernel-Supported Threads:
        - Dikelola oleh kernel sistem operasi, dan kernel aware tentang keberadaan thread ini.
        - Memungkinkan manajemen dan penjadwalan thread yang lebih baik, termasuk pemanfaatan multi-core CPU.
        - Meskipun lebih lambat dalam hal switching thread karena interaksi dengan kernel, memberikan dukungan yang lebih baik untuk multitasking dan paralelisme.

9. Tiga Model Multithreading:
    - Many-to-One Model: Banyak thread pengguna dipetakan ke satu thread kernel. Hanya satu thread yang dapat berjalan pada satu waktu, sehingga tidak memanfaatkan multi-core CPU.
    - One-to-One Model: Setiap thread pengguna dipetakan langsung ke satu thread kernel. Ini memungkinkan eksekusi thread secara bersamaan pada multi-core CPU dan mengurangi masalah terkait dengan many-to-one model.
    - Many-to-Many Model: Banyak thread pengguna dipetakan ke banyak thread kernel. Model ini memungkinkan fleksibilitas yang lebih tinggi dan dapat memanfaatkan multi-core CPU secara lebih efektif, dengan thread pengguna yang dapat ditingkatkan atau dikurangi sesuai kebutuhan.

10. State pada Java Thread:
    - New: Thread baru dibuat tetapi belum dijadwalkan untuk dieksekusi.
    - Runnable: Thread siap untuk dieksekusi dan berada dalam antrean thread. Ini termasuk thread yang sedang berjalan atau thread yang siap untuk dijadwalkan oleh scheduler.
    - Blocked: Thread sedang menunggu untuk mendapatkan akses ke sumber daya yang sedang digunakan oleh thread lain, misalnya, ketika mengakses monitor.
    - Waiting: Thread sedang menunggu untuk diberi notifikasi atau sinyal untuk melanjutkan eksekusi, biasanya menggunakan metode seperti wait().
    - Timed Waiting: Thread sedang menunggu untuk jangka waktu tertentu, menggunakan metode seperti sleep() atau join().
    - Terminated: Thread telah selesai eksekusi atau dihentikan. Setelah mencapai status ini, thread tidak akan kembali ke status lain.