Selesksi Bagian A Lab Sister 2022

13520163 Frederik Imanuel Louis

**A. Organisasi dan Arsitektur Komputer**

1. Define xor: a^b = (a&b) | (!a & !b)

a. Addition (LSB to MSB): (a+b) -> a_bit i + b_bit i + carry_i

Bit i sum = ((a_bit i ^ b_bit i) ^ carry_i)

Carry_i+1 = (a_bit i & (b_bit i | carry_i ) ) | (b_bit i & carry_i)

b. Subtraction (LSB to MSB): (a-b) -> a_bit i - b_bit i - carry_i

Bit i sum = ((a_bit i ^ b_bit i) ^ carry_i)

Carry_i+1 = (a_bit i & (b_bit i & carry_i ) ) | (!a_bit i & (b_bit i | carry_i))

c. Multiplication (bitwise): a*b = sum of (a_bit i * b_bit j)

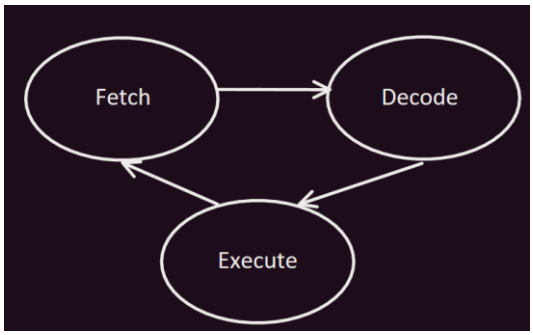
a_bit i * b_bit j = a_bit i & b_bit j

*note: summation done seperately

d. Division: use multiplication and substraction

2. Micro operation adalah operasi dasar yang dilakukan pada level register, misalnya shift, and, or, xor dll. Semua instruksi mesin kompleks diimplementasikan menggunakan rangkaian micro ops. Micro ops dilakukan secara fisik menggunakan implementasi microarsitektur CPU. Operasi yang sama dapat memiliki desain mikroarsitektur yang berbeda pada CPU yang berbeda.

3.




Fetch decode execute merupakan siklus yang dilakukan komputer untuk menjalankan instruksi. Fetch mengambil instruksi selanjutnya, dan mengubah instruction pointer ke instruksi berikutnya. Decode akan menerjemahkan instruksi menjadi instruksi yang dapat dijalankan, misalnya konversi address menjadi physical address. Execute akan menjalankan instruksi tersebut. Jika suatu interrupt diberikan, pada awal FDE cycle, komputer akan melayani interrupt tersebut (jika tidak ada interrupt dengan prioritas lebih tinggi yang sedang dijalankan). Saat komputer pertama kali dinyalakan, FDE cycle menjalankan bootloader yang terletak pada ROM. Siklus ini akan terus berjalan sampai komputer dimatikan.

4. Untuk mengatasi no exec bit, kita dapat memanggil shell dari fungsi libc (ret2libc), dengan memanggil system() dari libc. Tetapi karena terdapat randomization address space, kita perlu memanggil system@PLT dari procedure linkage table, yang posisinya tidak pernah dirandomize. Passing parameter "/bin/sh" juga dapat menggunakan strcpy@PLT. Tentu saja, semua pemanggilan fungsi tersebut menggunakan eksploitasi buffer overflow dari scan buf[256]. Dengan itu, kita cukup memasukkan payload pada program yang berisi: (1) 256 byte junk, (2) pengcopyan "/bin/sh" dengan beberapa kali pemanggilan strcpy@PLT, (3) pemanggilan "system@PLT", (4) noop sled secukupnya.

5. Memory dengan hierarchy yang lebih tinggi memiliki access time yang lebih rendah. Hal tersebut dapat dimanfaatkan untuk melakukan timing attack, misalnya dengan membuat page fault atau cache miss menggunakan input yang sesuai. Terlebih lagi, jika paging atau caching ini dipengaruhi oleh branching, maka value yang dibandingkan pada conditional jump dapat diketahui dari execution time yang bertambah cepat/lambat (akibat access time yang berbeda dari kedua branch).

6. a. Register renaming adalah pemisahan logical register dan physical register (seperti logical dan physical memory), dimana instruksi machine language akan diberikan dengan logical register, dan processor akan merubahnya menjadi physical register. Teknik ini menghilangkan dependensi dari penggunaan register secara berulang yang sebenarnya tidak memiliki dependensi data. Hal tersebut akan memunculkan lebih banyak pararelisme pada tingkat instruksi.

b. Instruction pipeline adalah teknik untuk mengimplementasi pararelisme pada level instruksi pada satu prosesor. Hal tersebut dapat dilakukan misalnya dengan membaca instruksi selanjutnya di tengah eksekusi suatu instruksi. Ini dapat dilakukan akibat cycle Fetch-Decode-Execute yang menggunakan bagian berbeda dari prosesor untuk tiap tahapannya.

c. Stack engine adalah alat yang secara otomatis menangani pemindahan stack pointer (rsp+8, rsp-8) saat dilakukan push/pop/ret dalam instruksi mesin. Hal tersebut mengurangi beban ALU yang sebelumnya bertanggung jawab untuk pemindahan stack pointer tersebut. Mengingat bahwa pemindahan stack pointer dilakukan tiap kali ada pemanggilan dan return fungsi, stack engine merupakan optimasi yang cukup signifikan.

**B. Sistem Operasi**

1. Zombie process adalah process yang sudah selesai melakukan seluruh instruksinya (sudah exit), tetapi masih tercatat di process table. Hal tersebut berarti proses tersebut masih PID (process ID) oleh operating system. Zombie process biasanya muncul akibat child process sudah exit, tapi tidak di-wait oleh parent processnya. Saat parent process tersebut masih berjalan dan child process sudah exit dan tidak di-wait, terciptalah zombie process. Jika parent process kemudian exit tanpa melakukan wait, child process biasanya akan "dibersihkan" secara otomatis oleh operating system, misalnya dengan mengassign parent dari process tersebut ke init (PID 1), dan initlah yang akan melakukan wait sehingga zombie process akan hilang. Reassign parent dari child process tersebut merupakan proses yang dilakukan fungsi *find_new_reaper*. Zombie process yang masih ada setelah parentnya sudah exit menandakan adanya bug dalam operating system.

2. Scheduler adalah proses yang mengalokasikan resource CPU kepada task yang berjalan pada suatu komputer. O(1) scheduler dan Completely Fair Scheduler menggunakan algoritma yang berbeda saat mengalokasikan resource tersebut. O(1) scheduler memprioritaskan kecepatan pengambilan keputusan

saat mengalokasikan resource tersebut (dengan waktu konstan) sehingga menghasilkan overhead yang lebih kecil. CFS memprioritaskan utilisasi CPU maksimal yang dibagi se-adil mungkin antar task yang sedang berjalan. Proses I/O bound adalah proses yang membutuhkan lebih banyak waktu melakukan I/O dibandingkan utilisasi CPU, dan sebaliknya, proses CPU bound membutuhkan lebih banyak utilisasi CPU dibandingkan I/O.

3. Virtual memory juga memiliki peran dalam memory protection. Page table yang diimplementasikan dalam virtual memory dapat mengassign tiap page dengan akses tertentu, misalnya read only, no execute, dsb. Memory protection dilakukan pada level hardware dan software. Dari segi software, operating system dapat memberikan proteksi misalnya dengan memberikan base dan limit register pada suatu proses. Dari segi hardware, MMU (dan dengan itu MPU), dapat memastikan bahwa memori yang diakses dilakukan menggunakan hak akses yang sesuai.

4. Jika command cd berhasil dieksekusi oleh shell, maka current working directory dari shell dan semua thread yang berjalan berubah menjadi directory yang ditunjuk parameter cd. Command cd yang digunakan merupakan fungsi built-in shell agar fungsi dari cd, yaitu merubah current working directory, dapat terlaksana dengan baik. Jika cd dieksekusi sebagai binary, maka current working directory yang akan diubah adalah current working directory dari lingkungan eksekusi cd, yaitu “subshell” yang dibuat oleh exec(). Kemudian, setelah subshell tersebut exit, cwd dari shell utama tidak akan berubah. Perlu dicatat bahwa /bin/cd tetap ada dan jika dijalankan, memiliki behaviour yang disebutkan diatas.

5. Copy on Write membuat copy dari suatu data pada memory hanya jika data tersebut ingin dimodifikasi (write). Hal tersebut mempercepat fork/clone karena proses child dan parent dapat merujuk ke datayang sama dalam memory tanpa perlu pengcopyan seluruh page yang digunakan program tersebut jika tidak ada data yang berubah. Hanya segmen data yang perlu diubah yang akan dicopy pada memory.

6. Sebuah komputer secara garis besar terbagi menjadi dua bagian, yaitu hardware dan software. Hardware adalah perangkat keras yang dapat dilihat secara fisik pada komputer, misalnya CPU, monitor, dan mouse. Software adalah perangkat lunak yang bekerja di dalam komputer, yang dapat dilihat sebagai “otak” dari komputer. Fungsi-fungsi komputer yang sering digunakan, seperti bermain game, menonton youtube, menulis dokumen, dilakukan dengan software yang sesuai. Operating system adalah software dasar yang menjembatani komunikasi antara hardware dan seluruh software lainnya. Seluruh instruksi yang dikirimkan software tingkat tinggi, misalnya menonton video, akan “diterjemahkan” oleh operating system menjadi instruksi yang dapat dijalankan oleh hardware. Komunikasi dari hardware ke software, misalnya gerakan dan klik mouse, juga dilakukan melalui operating system.

Operating system melakukan hal tersebut dengan menggunakan Basic Input Output System (BIOS) yang disediakan oleh hardware untuk melakukan operasi yang diperlukan software lainnya. Operasi tersebut disediakan melalui kernel dan dapat diakses secara langsung melalui shell. Selain itu, saat komputer dinyalakan, operating system adalah software pertama yang akan diload kedalam memory menggunakan bootloader yang terdapat pada Read Only Memory (ROM).

**C. Jaringan Komputer**

1. Dynamic IP adalah IP address yang diassign pada user yang dapat berubah dari waktu ke waktu, sedangkan Static IP adalah IP address yang tidak pernah berubah. Static IP diberikan oleh Internet Service Provider (ISP) sedangkan Dynamic IP diberikan oleh Dynamic Host Configuration Protocol (DHCP). Static

IP digunakan jika ada third party yang perlu mengingat IP address user, seperti untuk whitelist IP. Dynamic IP lebih umum digunakan karena sifatnya lebih aman. Dynamic IP lebih sulit dilacak ke pengguna, sedangkan Static IP dapat langsung dilacak ke pengguna.

2. ARP Poisoning adalah teknik mengalihkan semua traffic dari dan menuju suatu IP Adresss ke attacker. Hal tersebut dilakukan dengan memberi ARP yang spoofed ke suatu network agar MAC (physical) address dari device attacker diasosiasikan dengan IP dari user yang ingin diserang. Koneksi tersebut dapat kemudian digunakan dalam man in the middle attack.

3. TCP merupakan protokol koneksi yang memungkinkan komunikasi dua arah antara client dan server setelah three-way handshake. Pertama-tama, client mengirimkan segmen dengan SYN (Synchronize Sequence Number) kepada server yang menandakan client ingin memulai komunikasi dengan server. Kemudian, server membalas client dengan signal SYN-ACK, dan terakhir , client mengirimkan kembali signal ACK kepada server dan transfer data akan dimulai.

4. Programming dengan jaringan akan melakukan komunikasi dengan proses lain, biasanya pada mesin yang berbeda. Komunikasi ini harus dilakukan dengan protokol yang tepat dan memerhatikan aspek keamanan dari komunikasi. Data yang diterima maupun dikirim harus dipastikan aman dan jika diperlukan, rahasia. Programming biasa (diasumsikan tanpa jaringan) umumnya lebih fokus pada aspek yang berbeda, misalnya use case pengguna, efisiensi waktu dan memori program, dan sebagainya.

5. Asumsi: komunikasi dilakukan dengan node terkecil (secara alfabet) yang belum pernah ditemui dan komunikasi pada tiap tahap dilakukan secara bersamaan.

Tahap 1:

| | Node A | | | | | Node B | | | | | Node C | | | | | Node D | | | | | Node E | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | A | B | C | D | E | A | B | C | D | E | A | B | C | D | E | A | B | C | D | E | A | B | C | D | E |
| A | 0 | 7 | - | - | 1 | | | | | | | | | | | | | | | | | | | | |
| B | | | | | | 7 | 0 | 1 | - | 8 | | | | | | | | | | | | | | | |
| C | | | | | | | | | | | - | 1 | 0 | 2 | - | | | | | | | | | | |
| D | | | | | | | | | | | | | | | | - | - | 2 | 0 | 2 | | | | | |
| E | | | | | | | | | | | | | | | | | | | | | 1 | 8 | - | 2 | 0 |

Tahap 2:

| | Node A (+B) | | | | | Node B (+A) | | | | | Node C (+B) | | | | | Node D (+C) | | | | | Node E (+A) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | A | B | C | D | E | A | B | C | D | E | A | B | C | D | E | A | B | C | D | E | A | B | C | D | E |
| A | 0 | 7 | 8 | - | 1 | 0 | 7 | 8 | - | 1 | | | | | | | | | | | 0 | 7 | - | 3 | 1 |
| B | 7 | 0 | 1 | - | 8 | 7 | 0 | 1 | - | 8 | 7 | 0 | 1 | 3 | 8 | | | | | | | | | | |
| C | | | | | | | | | | | 8 | 1 | 0 | 2 | 9 | - | 1 | 0 | 2 | 4 | | | | | |
| D | | | | | | | | | | | | | | | | - | 3 | 2 | 0 | 2 | | | | | |
| E | | | | | | | | | | | | | | | | | | | | | 1 | 8 | - | 2 | 0 |

Tahap 3:

| | Node A (+C) | Node B (+C) | Node C (+A) | Node D (+B) | Node E (+B) |
|---|---|---|---|---|---|

| | A | B | C | D | E | A | B | C | D | E | A | B | C | D | E | A | B | C | D | E | A | B | C | D | E |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | 0 | 7 | 8 | 10 | 1 | 0 | 7 | 8 | 10 | 1 | 0 | 7 | 8 | 10 | 1 | 0 | 7 | 8 | 10 | 1 | 0 | 7 | 8 | 3 | 1 |
| B | 7 | 0 | 1 | 3 | 8 | 7 | 0 | 1 | 3 | 8 | 7 | 0 | 1 | 3 | 8 | 7 | 0 | 1 | 3 | 5 | 7 | 0 | 1 | 10 | 8 |
| C | 8 | 1 | 0 | 2 | 4 | 8 | 1 | 0 | 2 | 4 | 8 | 1 | 0 | 2 | 4 | 8 | 1 | 0 | 2 | 4 | | | | | |
| D | | | | | | | | | | | | | | | | 10 | 3 | 2 | 0 | 2 | | | | | |
| E | | | | | | | | | | | | | | | | | | | | | 1 | 8 | 9 | 2 | 0 |

Tahap 4:

| | Node A (+D) | | | | | Node B (+D) | | | | | Node C (+D) | | | | | Node D (+A) | | | | | Node E (+C) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | A | B | C | D | E | A | B | C | D | E | A | B | C | D | E | A | B | C | D | E | A | B | C | D | E |
| A | 0 | 7 | 8 | 10 | 1 | 0 | 7 | 8 | 10 | 1 | 0 | 7 | 8 | 10 | 1 | 0 | 7 | 8 | 10 | 1 | 0 | 6 | 5 | 3 | 1 |
| B | 7 | 0 | 1 | 3 | 5 | 7 | 0 | 1 | 3 | 5 | 7 | 0 | 1 | 3 | 5 | 7 | 0 | 1 | 3 | 5 | 6 | 0 | 1 | 3 | 5 |
| C | 8 | 1 | 0 | 2 | 4 | 8 | 1 | 0 | 2 | 4 | 8 | 1 | 0 | 2 | 4 | 8 | 1 | 0 | 2 | 4 | 5 | 1 | 0 | 2 | 4 |
| D | 10 | 3 | 2 | 0 | 2 | 10 | 3 | 2 | 0 | 2 | 10 | 3 | 2 | 0 | 2 | 10 | 3 | 2 | 0 | 2 | 3 | 3 | 2 | 0 | 2 |
| E | | | | | | | | | | | | | | | | | | | | | 1 | 5 | 4 | 2 | 0 |

Tahap 5:

| | Node A (+E) | | | | | Node B (+E) | | | | | Node C (+E) | | | | | Node D (+E) | | | | | Node E (+D) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | A | B | C | D | E | A | B | C | D | E | A | B | C | D | E | A | B | C | D | E | A | B | C | D | E |
| A | 0 | 6 | 5 | 3 | 1 | 0 | 6 | 5 | 3 | 1 | 0 | 6 | 5 | 3 | 1 | 0 | 6 | 5 | 3 | 1 | 0 | 6 | 5 | 3 | 1 |
| B | 6 | 0 | 1 | 3 | 5 | 6 | 0 | 1 | 3 | 5 | 6 | 0 | 1 | 3 | 5 | 6 | 0 | 1 | 3 | 5 | 6 | 0 | 1 | 3 | 5 |
| C | 5 | 1 | 0 | 2 | 4 | 5 | 1 | 0 | 2 | 4 | 5 | 1 | 0 | 2 | 4 | 5 | 1 | 0 | 2 | 4 | 5 | 1 | 0 | 2 | 4 |
| D | 3 | 3 | 2 | 0 | 2 | 3 | 3 | 2 | 0 | 2 | 3 | 3 | 2 | 0 | 2 | 3 | 3 | 2 | 0 | 2 | 3 | 3 | 2 | 0 | 2 |
| E | 1 | 5 | 4 | 2 | 0 | 1 | 5 | 4 | 2 | 0 | 1 | 5 | 4 | 2 | 0 | 1 | 5 | 4 | 2 | 0 | 1 | 5 | 4 | 2 | 0 |

6. Dalam jaringan, topologi digunakan untuk memodelkan hubungan antar elemen-elemen jaringan. Secara umum, terdapat dua jenis topologi jaringan, yaitu topologi fisik, dan topologi logika. Topologi fisik memodelkan lokasi dan metode penghubungan (cabling) tiap elemen, sedangkan topologi logika memodelkan bagaiman signal dan data dikirimkan dari satu elemen ke elemen lainnya.

7. Data bit = 26, Redundant bit = 5 -> 2^5 = 26 +5 +1. R -> redundant bit, D-> data bit

Pembuatan data: (even parity)

```
import random

def calculateRedundant(size):
    i = 0
    while(i**2 < i + size +1):
        i+=1
    return i

def randomBit():
    return chr(ord('0') + random.randint(0,1))
```

```
def createRandomHamming(size):
    redundant = calculateRedundant(size)
    power = 0
    ret = ([])
    data = ''
    for i in range(size+redundant):
        if (2**power == i):
            ret.insert(0,'0')
            power+=1
        else:
            temp = randomBit()
            ret.insert(0, temp)
            data = temp + data
    for i in range(redundant):
        num = 2**i
        count = 0
        for j in range(1, size+1):
            if num&j != 0 and ret[len(ret) - j] == '1':
                count += 1
        if (count%2 == 1):
            ret[len(ret) - num] = '1'
    return (data, ''.join(ret))

if __name__ == '__main__':
    data, hamming = createRandomHamming(26)
    print(data)
    print(hamming)
```

Data: 11100010001101010111001101101

Data + Redundant (no noise):  11100010001101001110011011100011

Noise: LSB ke 3 -> 11100010001101001110011011100111

Koreksi:

Total bit set 1: ganjil

Total bit set 10: ganjil

Total bit set 100: genap

Total bit set 1000: genap

Total bit set 10000: genap

Bit error: 11 (LSB ke 3)

**D. Sistem Pararel dan Terdistribusi**

1. a. MPI_INT menandakan bahwa MPI akan memperlakukan isi dari memori yang ditandai dengan MPI_INT sebagai integer. Primitive type int menandakan bahwa suatu variabel bertipe integer. MPI_INT tidak dapat digunakan untuk menginisialisasi variabel.

b. Directive parallel menandakan bahwa code dapat dieksekusi oleh beberapa thread secara parallel, dan atomic menandakan bahwa suatu address memory harus diupdate secara atomik. Parallel memungkinkan eksekusi proses yang lebih cepat dengan beberapa thread, tetapi atomic meningkatkan keamanan data. Parallel dapat membawa resiko seperti reader writer problem yang muncul akibat akses data secara bersamaan oleh beberapa thread.

c. MPI_Reduce mengambil hasil dari semua proses dan mengeluarkan hasil yang tereduksi ke proses root. Reduksi dapat menggunakan berbagai fungsi, misalnya MPI_SUM. MPI_Allreduce melakukan hal yang sama, tetapi mengirimkan hasil tereduksi tersebut ke semua proses, bukan hanya root.

2. Profiler GPU dapat menggambarkan interaksi CPU dan GPU, serta timeline eksekusi algoritma pada GPU (dan CPU). Dengan profiler, kita dapat mengecek bagaimana utilisasi CPU dan GPU, apakah CPU banyak menunggu GPU, apakah GPU banyak menunggu CPU, dan sebagainya. Dari informasi tersebut, dapat dilakukan optimisasi, misalnya mencari peluang konkurensi yang terlewatkan. Misalnya, pada algoritma paralel GPU, sering kali salah satu bottleneck berada di proses memcpy atau memindahkan data ke GPU. Untuk mengoptimasi hal tersebut, kita dapat mencari cara untuk meningkatkan utilisasi GPU ketika memcpy tersebut dilakukan, misalnya dengan overlapping data transfer dan komputasi data lainnya.

3. GPU pada dasarnya adalah CPU kemampuan rendah dengan ratusan hingga ribuan core. Oleh karena itu, multi CPU lebih bagus digunakan untuk melakukan proses yang membutuhkan banyak akses memory, i/o, atau multitasking (context switch). Multi GPU lebih bagus digunakan untuk melakukan proses dengan parallelism yang tinggi, misalnya proses yang memerlukan banyak perhitungan (yang biasanya bisa dikerjakan secara paralel). Proses tersebut termasuk proses grafika, training AI, dsb.

4. There is one administrator fallacy adalah miskonsepsi bahwa hanya ada satu administrator dari suatu sistem. Makin besar suatu sistem, makin banyak orang yang akan terlibat dalam pengembangan dan penggunaan sistem tersebut. Jika sistem sudah terlalu besar, tidak mungkin ada satu orang administrator yang dapat mengetahui segalanya mengenai sistem tersebut. Selain itu, pada sistem kecil, satu administrator yang mengetahui segalanya juga tidak selamanya dapat merawat dan menjalankan sistem tersebut. Fallacy ini mengingatkan developer bahwa diperlukan dokumentasi dan backup yang baik dari sebuah sistem, agar sistem tersebut dapat dikembangkan dengan baik.

5. Tiap clock dalam tiap sistem berjalan dengan rate yang berbeda, sehingga mereka tidak akan sinkron. Hal tersebut dapat menyebabkan penjadwalan proses, pengurutan event, debugging, dsb menjadi sulit karena waktu yang tercatat untuk tiap sistem belum tentu sinkron.

6. Konsensus Paxos menggunakan Generation Clock yang merupakan bagian dari konsensus Leaders and Followers, dimana tiap node memiliki Generation Clocknya masing-masing (tidak akan dijelaskan disini). Konsensus Paxos juga menggunakan konsep Quorum, dimana diperlukan persetujuan dari mayoritas node untuk melakukan perubahan. Konsensus Paxos secara garis besar terbagi menjadi tiga tahap, yaitu Prepare, Accept, dan Commit.

Fase Prepare dimulai ketika satu atau lebih node dalam sistem menerima request, dan ingin melakukan suatu perubahan (misalnya mengubah data). Node yang menerima request tersebut dinamakan node Proposer, dan semua node (termasuk Proposer), dinamakan node Acceptor. Semua proposer akan mengirim Prepare Request ke Acceptor satu per satu. Prepare Request terdiri dari Generation Number yang diperoleh dari Generation Clock Proposer, dan ID Proposer. Acceptor akan mengeluarkan response Promise jika Prepare Request yang diterima memiliki Generation Number yang lebih besar dari Prepare Request lainnya yang sudah diterima (jika ada). Selain itu, jika Acceptor sudah pernah menerima Accept Request (pada fase Accept) yang belum dicommit, maka value (data yang ingin diubah) Proposer akan berubah mengikuti value dari Accepted Request jika Accept Request tersebut memiliki Generation Number yang lebih tinggi. Jika Proposer gagal mendapatkan Quorum setelah mengirimkan Prepare Request ke seluruh Acceptor, maka Generation Clock Proposer akan diincrement, dan Proposer akan mengulang fase Prepare ini dari awal.

Fase Accept dimulai jika node Proposer sudah memilki Quorum, yaitu jika mayoritas Acceptor sudah memberikan Promise pada Proposer. Proposer kemudian akan mengirimkan Accept Message yang berisi Generation Number dan ID Proposer, serta value yang ingin diubah. Accept Message akan dikirimkan pada Acceptor yang sudah memberi Promise pada fase Prepare. Acceptor akan menerima Accept Request jika promise terakhir yang ia berikan adalah ke Proposer tersebut. Jika ia telah memberikan promise baru ke Proposer lain, Acceptor tersebut akan menolak Accept Request tersebut. Jika Proposer kehilangan Quorum (ada Accept Request yang direject), maka Generation Clock Proposer akan diincrement, dan Proposer akan mengulang proses dari fase Prepare.

Fase Commit akan dimulai jika Proposer sudah mendapatkan Quorum dari Fase Accept (mayoritas node sudah menerima Accept Request). Propser akan kemudian mengirim commit message ke semua Node, dan kemudian data akan diubah ke value yang dimiliki proposer. Perhatikan bahwa value ini belum tentu sama dengan value yang awalnya diterima Proposer dari request user (value dapat berubah pada fase prepare).