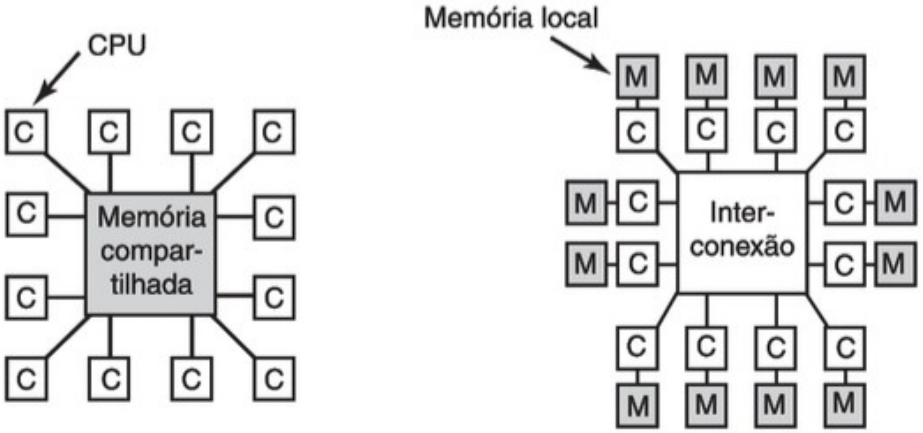
Universidade de Caxias do Sul
Área do Conhecimento de Ciências Exatas e Engenharias
CIC4004A - Programação Concorrente, Paralela e Distribuída
Professor: André Luis Martinotto
Primeira Avaliação

QUESTÃO 1) (1,5 PONTOS) Na figura abaixo, retirada do livro "Sistemas Operacioanis Modernos" de Andrew Tanenbaum, tem-se diferentes exemplos de arquiteturas paralelas.

a) Identifique a qual categoria da taxonomia de Flynn pertencem as arquiteturas abaixo. Justique sua resposta.

b) Quais dessas arquiteturas paralelas podem ser classificadas com um sistema distribuído? Por quê?



QUESTÃO 2) (1,0 PONTO) Em um grande sistema distribuído é comum a existência de vários tipos de arquiteturas de computadores, cada uma delas com sua própria representação para números, caracteres e outros dados. Essas diferenças podem gerar empecilhos inviabilizando uma comunicação harmoniosa entre os sistemas. Em RPC a solução para esse problema consiste em utilizar uma representação independente de arquitetura da máquina (comumente XDR). Sempre que for estabelecida uma comunicação os "stubs" convertem os dados nativos para esse tipo de representação (XDR). Pergunta-se: em RMI é necessário que os "stubs" do cliente e do servidor convertam formatos de dados, como é necessário em RPC?

QUESTÃO 3) (1,5 PONTOS) Seja um sistema de controle em tempo-real de uma usina nuclear. O sistema é composto de diversos sensores que monitoram pressão, temperatura, nível de radiação etc. e enviam os dados para um controlador. O controlador, conforme o valor informado, define as ações de controle (por exemplo, abrir ou fechar determinados registros até determinado ângulo) e as envia para um atuador, que realiza tais ações. Observe que todo o processo tem que ser extremamente rápido, ou seja, as tarefas têm prazos ("deadlines"). Se os dados dos sensores chegarem no controlador fora de um prazo pré-determinado, eles são ignorados. Analogamente, se as ações de controlador, chegam no atuador fora de prazo, elas também são ignoradas. Supondo que você fosse projetar tal sistema utilizando como mecanismo de comunicação soquetes, qual tipo de soquete você utilizaria (transmissão de datagramas via TCP/IP, ou de "streams" via UDP/IP) para cada canal de comunicação (sensores/controlador, controlador/atuador)? Justifique, considerando as características da aplicação e do protocolo de comunicação.

QUESTÃO 4) Dado uma biblioteca de comunicação com bufferização (mensagens não bloqueantes enviadas e não recebidas ainda são guardadas automaticamente pelo sistema operacional) que apresenta as seguintes primitivas:

- send (<msg> , <id>): versão não bloqueante para o envio de uma mensagem. A primitiva send envia uma mensagem contida em <msg> para o processo identificado por <id>.
- bsend (<msg>, <id>): versão bloqueante para o envio de uma mensagem. A primitiva bsend envia uma mensagem contida em <msg> para o processo identificado por <id>.
- recv (<msg>, <id>): recebe uma mensagem do processo <id> e armazena no endereço <var> . A primitiva recv é bloqueante.

Analise a possibilidade de ocorrência de bloqueio (deadlock) para as programações listadas abaixo listadas. Considere que entre as invocações das primitivas de troca de mensagens podem existir sequências de instruções normais (Sugestão: faça um diagrama representando o envio e o recebimento das mensagens)

a) (0,5 PONTO) Os processos abaixo estão em uma situação de deadlock? Porquê? **Observação:** O caracter * índica que se deseja receber uma mensagem de qualquer processo.

Processo 0	Processo 1	Processo 2

int main(){
send (msg, 1);
recv (msg, 1);
recv (msg, 1);
send (msg, 1);
send (msg, 2);
send (msg, 1);
recv (msg, *);
}
	int main(){
	recv (msg, 0);
	send (msg, 2);
	send (msg, 0);
	send (msg, 0);
	recv (msg, 0);
	send (msg, 0);
	recv (msg, 0);
	}
		int main(){
		recv (msg, 0);
		recv (msg, 1);
		}

b) (0,5 PONTO) Os processos abaixo estão em uma situação de deadlock? Porquê? **Observação:** os processos abaixo são *daemons*!!!

Processo 0	Processo 1

int main(){
 	
while(TRUE) {
...
recv(msg, 1);
...
send(msg , 1);
}
}
	#define N 5
	int main(){
	for (i = 0; i < N; i++)
	send(msg, 0);
	while(TRUE) {
	...
	recv(msg , 0);
	...
	send(msg , 0);
	}
	}

c) (0,5 PONTO) Os processos abaixo estão em uma situação de deadlock? Porquê?

Processo 0	Processo 1	Processo 2

int main(){
bsend (msg, 1);
recv (msg, 2);
}
	int main(){
	bsend (msg, 2);
	recv (msg, 0);
	}
		int main(){
		send (msg, 0);
		recv (msg, 1);
		}

QUESTÃO 5) Seja um programa cliente que para calcular o coeficiente binomial chama um procedimento remoto chamado fatorial, que é executado em uma máquina referenciada como MAQ_SERV, conforme os códigos abaixo que omitem, por questões de simplicidade, uma série de aspectos relacionados a RPC.

CLIENTE	SERVIDORA

#include<rpc.h>
 	
void main(){
long int, fatn, fatp, fatnp;
int n, p;
double coef;
printf("Entre com n e p\n");
scanf("%d %d", &n, &p);
rpc_call(MAQ_SERV, fatorial,n,fatn);
rpc_call(MAQ_SERV, fatorial,p,fatp);
rpc_call(MAQ_SERV, fatorial(n-p),fatn_p);
coef = (double)fatn/(fatp*fatn_p);
printf("Coeficiente binomila %f\n", coef);
}
	long int fatorial(int k){
	long int f = 1;
	int i;
	for (i=1; i<=k; i++){
	f*= i;
	}
	return f;
	}

a) (1,0 PONTO) Reescreva o códigos do programa acima para que as relações cliente-servidor sejam estabelecidas através de sockets TCP.

b) (1,0 PONTO) Reescreva o códigos do programa acima para que as relações cliente servidor sejam estabelecidas através de sockets UDP.

Para desenvolvimento utilize uma linguagem de programação real e considere uma biblioteca de *sockets* que apresenta as seguintes primitivas:

sock = socket(Protocolo):	Criação de um *socket*. Protocolo TCP ou UDP;
bind(socket, porta):	Estabelece uma porta ao *socket;*
listen(socket, quantos):	Definição do tamanho da lista de requisições;
connect (socket, ip, porta):	Requisição de uma conexão;
new_sock = accept (socket):	Espera por uma conexão. Retorna um *socket* que será usado para a comunicação;
send (socket, dados):	Envia uma mensagem com o conteúdo de *dados;*
sendto (socket, dados, end):	Envia uma mensagem com o conteúdo de *dados* para o destinário *end.* Utilize SERVIDOR para referenciar o endereço IP e porta da tarefa servidora;
recv(socket, dados):	Recebe uma mensagem e armazena em *dados;*
recvfrom(socket, dados, end)	Recebe uma mensagem armazendo seu conteúdo em *dados* e captura o endereço emissor em *end;*
close(socket):	Fecha a conexão;

QUESTÃO 6) (2,5 PONTOS) O trecho de código abaixo calcula a variância e o desvio padrão para um vetor amostra[N].

```
soma = 0;
for (i=0;i<N;i++){
        soma += amostra[i];
}
media = soma/N;

soma = 0;
for (i=0;i<N;i++){
        soma += (amostra[i]-media) * (amostra[i]-media);
}

variancia = soma/(N-1);
desvio = sqrt(variancia);
```

Altere o trecho de código acima de forma de forma que o cálculo seja efetuado por P processos. O processo pai também deve efetuar parte do processamento. Para o desenvolvimenot utilize uma biblioteca de troca de mensagens que segue o modelo SPMD (do inglês *single program, multiple data*) e com as primitivas listadas abaixo. Destaca-se que tanto o cálculo da média, como o cálculo do desvio padrão, devem ser realizados em paralelo pelos *P* processos.

int rank():	Retorna o identificador de cada processo(de 0 a N-1);
int size():	Retorna o número de processos;
send(<var>,<id>):	Envia uma mensagem contida em <var>para a tarefa identificada por <id> .
receive(<var>,<id>):	Recebe uma mensagem do processo <id> e armazena na variável <var>

Prova 1 - Solução

/*----------*/

Questão 1)

a) Ambas as arquiteturas correspondem a arquiteturas de categoria MIMD (*Multiple Instruction Multiple Data*), onde múltiplas instruções são executadas, e cada uma delas operando sobre dados independentes. De fato, essas arquiteturas correspondem, respectivamente, a computadores multiprocessados (multicore) e clusters de computadores.

b) A primeira arquitetura (da esquerda) não corresponde a um sistema distribuído, uma vez que essa apresenta uma memória compartilhada e um relógio global. Já a segunda arquitetura corresponde a um sistema distribuído uma vez que cada componente é independente, apresentado sua própria memória e seu próprio relógio.

/*----------*/

Questão 2)

Em RMI não é necessário que os “stubs” do cliente e do servidor convertam os dados para uma formato padrão, uma vez que os dados são transmitidos no formato da máquina virtual Java (bytecodes) que independe de arquitetura.

/*----------*/

Questão 3)

O protocolo UDP é mais leve e rápido do que o TCP, uma vez que o protocolo UDP não necessita de confirmação de envio de resposta. Já, o protocolo TCP necessita receber resposta de envio e conforme a situação ele ainda retransmitirá o pacote. Além disso, o UDP possui um cabeçalho menor.

Desta forma, o protocolo UDP é mais adequado em ambos os casos, uma vez que a transmissão não admite retardo. Além disso, em caso de retardo os pacotes são simplesmente ignorados. Desta forma, as perdas que o protocolo UDP possui são menos importantes do que o atraso.

/*----------*/

Questão 4)

a) Não estão bloqueados. Todos os processos terminam sua execução corretamente. Abaixo tem-se um fluxo de execução dos mesmos.

Processo 0	Processo 1	Processo 2
int main(){	int main(){	int main(){
send (msg, 1);	recv (msg, 0);	
	send (msg, 2);	(msg,1) - Bufferizada
recv (msg, 1);	send (msg, 0);	
recv (msg, 1);	send (msg, 0);	Lê do buffer
send (msg, 1);	recv (msg, 0);	
send (msg, 2);		recv (msg, 0);
send (msg, 1);	send (msg, 0);	recv (msg, 1);
Lê do buffer recv (msg, *);	recv (msg, 0); Lê do buffer	
}	}	}

b) Não estão bloqueados. É a implementação clássica do Produtor-consumidor com troca de mensagens. Neste caso, o processo 1 envia N mensagens para o processo 0, sendo que as mensagens enviadas são bufferizadas. No momento que o processo 0 receber uma mensagem (recv), ele enviará uma nova mensagem (send) para o processo 1, sendo que o processo 1 receberá essa mensagem e enviará uma nova mensagem para o processo 0. Ou seja, no momento que o processo 0 e o processo 1 estiverem no laço eles ficaram alternando o envio e o recebimento de mensagens. Abaixo tem-se um fluxo de execução dos mesmos.

Processo 0	Processo 1
int main(){	#define N 5
Lê do buffer	int main(){
(msg,1) - Bufferizadas	for (i = 0; i < N; i++)
	send(msg, 0);
while(TRUE) {	
...	while(TRUE) {
recv(msg, 1);	...
...	recv(msg , 0);
send(msg , 1);	...
}	send(msg , 0);
}	}
	}

c) Não estão bloqueados. O processo 2 envia uma mensagem para o processo 0 e, posteriormente, executa o recv liberando o processo 1, que estava bloqueado no send bloqueante. Após, o processo 1 executa o recv liberando o processo 0, que estava bloqueado. Por fim, o processo 0 executa o recv lendo do buffer os dados enviandos pelo processo 2.

Processo 0	Processo 1	Processo 2
int main(){	int main(){	int main(){
bsend (msg, 1);	bsend (msg, 2);	send (msg, 0);
Lê do buffer recv (msg, 2);	recv (msg, 0);	recv (msg, 1);
}	}	}